IL BASIC PER TUTTI

Paul M. Chirlian

EDIZIONE
ITALIANA

GRUPPO
EDITORIALE
JACKSON

IL BASIC PER TUTTI

di
Paul M. Chirlian

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

SOMMARIO

PREFAZIONE

Questo libro è un testo propedeutico al linguaggio di programmazione BASIC. È destinato agli studenti degli istituti superiori e universitari che non hanno ancora familiarità con la programmazione, e più generalmente con i calcolatori.

Gli argomenti sono esposti in forma elementare, in modo che il lettore alle prime armi non si troverà in difficoltà. Comunque il libro è esauriente, per cui il lettore può acquisire familiarità con tutti gli aspetti del BASIC. Oltre ad esporre il linguaggio BASIC, sono esaminate le procedure necessarie per eseguire programmi scritti in BASIC: si fa riferimento perciò all'elaborazione a divisione di tempo (*timesharing*) e a quella a lotti (*batch processing*).

Gli argomenti sono presentati in modo che lo studente possa cominciare subito ad eseguire programmi in BASIC, ed acquistare quindi un'esperienza pratica che lo agevolerà nell'apprendimento.

Chiunque abbia pratica di programmazione sa che la messa a punto (*debugging* in inglese) dei programmi è un momento molto importante della programmazione. Nel paragrafo 1-3 sono presentati i concetti iniziali, mentre nel Capitolo 9 è fatta una trattazione più dettagliata della messa a punto di un programma. È importante che il programmatore non si scoraggi per la presenza di errori nel programma; a questo fatto è data particolare enfasi esaminando il problema della messa a punto.

Tutti i programmi complessi sono accompagnati dai diagrammi di flusso, nei quali è usata la notazione standard.

Nel paragrafo 4-5 è esaminato l'importante argomento della documentazione. Comprenderne l'importanza è fondamentale sia per i programmatori alle prime armi, sia per gli esperti.

Quindi si può dire che il libro tratta del linguaggio di programmazione BASIC in modo abbastanza esauriente: a lettura finita, al principiante saranno familiari quasi tutti i suoi aspetti.

Ci auguriamo che questo libro sia di aiuto anche per quei programmatori esperti che vogliano riesaminare questo o quell'aspetto del BASIC. Per facilitare la loro ricerca, le Appendici A e B presentano un glossario dei termini, delle espressioni e delle operazioni del BASIC, ed un elenco delle funzioni di libreria. L'Appendice C ri-

porta i comandi di sistema, mentre l'Appendice D tratta di alcune modifiche del linguaggio BASIC.

Al termine di ogni capitolo si troveranno molti e svariati esercizi; per gli insegnanti è disponibile un manuale con le soluzioni dei problemi.

RINGRAZIAMENTI

La mia più affettuosa gratitudine e i più sinceri ringraziamenti a mia moglie Barbara, che non solo mi ha aiutato con i suoi continui incoraggiamenti, e mi ha permesso di fare il mio lavoro tranquillamente, ma ha anche battuto a macchina la prima stesura e la versione definitiva del manoscritto e ha corretto il testo dal punto di vista formale. Inoltre mi sono state di grande aiuto le sue molte osservazioni editoriali.

Ringrazio affettuosamente anche mia figlia Lisa, che ha contribuito alla battitura ed alla correzione formale del manoscritto, e mio figlio Peter che innanzitutto mi ha dato l'idea di scrivere questo libro, e poi ha controllato alcune parti del manoscritto.

Paul M. Chirlian

CAPITOLO 1

INTRODUZIONE ALLA PROGRAMMAZIONE IN BASIC

Il moderno calcolatore esegue in pochi minuti calcoli che, fatti a mano, richiederebbero giorni o settimane. I suoi impieghi sono vari: aiuta il medico nelle diagnosi, è un valido strumento per prendere decisioni complesse come quella dell'appropriata collocazione di una fabbrica, o per ottenere una serie di grandezze statistiche da una massa di dati.

Ma c'è una cosa: il calcolatore non pensa. In ogni calcolo o apparente scelta logica è guidato da un programma scritto da un programmatore. Anche quando il risultato del calcolo o la scelta logica è il prodotto finale di una quantità enorme di operazioni che il calcolatore esegue molto rapidamente, questi calcoli sono sempre guidati da un programma. Questo libro si propone di insegnare a programmare i calcolatori.

Un programma consiste in una serie di istruzioni che guidano nel senso voluto l'attività del calcolatore. Per far questo occorre fissare un linguaggio che permetta al programmatore di istruire il calcolatore. Alcuni linguaggi, detti linguaggi di programmazione, sono di facile uso; fra questi vi è il BASIC, che unisce ad una notevole efficacia un'estrema facilità di apprendimento. È questo linguaggio l'argomento di questo libro.

1-1. IL CALCOLATORE NUMERICO

Inizieremo molto presto a scrivere programmi completi in BASIC, ma prima occorre illustrare brevemente le varie parti di un calcolatore numerico.

Le operazioni aritmetiche quali addizione e sottrazione sono eseguite nel *processore centrale*, che esegue anche operazioni logiche come verificare l'uguaglianza fra due numeri.

La *memoria principale* è utilizzata per contenere i dati di un programma, o anche

il programma in elaborazione. Questa memoria può memorizzare e fornire informazioni molto rapidamente (in un milionesimo di secondo, o anche meno). La dimensione di una memoria indica la quantità di dati che può contenere. Programmi complessi spesso richiedono una grande memoria principale.

La memoria principale è costosa, e quindi è limitata come dimensione. Spesso è necessario memorizzare grandi quantità di dati: ad esempio, una banca deve memorizzare tutte le informazioni riguardanti migliaia di clienti. Comunque, dato che le elaborazioni vengono eseguite su un conto per volta, sarebbe antieconomico memorizzare i dati di tutti i clienti nella memoria principale. I calcolatori sono perciò dotati di *dispositivi ausiliari di memorizzazione*, capaci di memorizzare dati in modo relativamente economico. Un esempio è rappresentato dai *dispositivi a nastro magnetico*, in cui le informazioni sono memorizzate su un nastro magnetico, che è simile a quelli usati in un comune registratore. Altri dispositivi sono i dischi e i cilindri magnetici. Tornando all'esempio fatto prima, tutti i dati riguardanti i clienti di una banca possono essere memorizzati su un nastro magnetico; quando poi si devono fare delle elaborazioni su un particolare conto, solo i dati relativi a quest'unico conto saranno trasferiti dal nastro nella memoria principale del calcolatore. Ad elaborazione terminata, i nuovi dati saranno riportati sul nastro.

Su nastro magnetico, oltre ai dati, si possono memorizzare anche programmi: se ad esempio un grosso programma dev'essere eseguito solo in certi casi, questo può essere memorizzato su nastro (o disco, etc.), da cui è letto, quando si presenta l'occasione di eseguirlo, nella memoria principale. Da questa è poi rimosso ad esecuzione avvenuta. In questo modo si evita che grosse porzioni della memoria principale siano occupate da informazioni che al momento non sono utilizzate.

Dati e programmi devono essere forniti al calcolatore, mentre i risultati dell'elaborazione all'utilizzatore: a questi due momenti provvedono i *dispositivi di ingresso/uscita*. Uno di questi dispositivi è la *telescrivente*, che è una specie di macchina da scrivere. Quando il programmatore batte un tasto della telescrivente, vengono generati dei segnali elettrici che mandano l'informazione al calcolatore; inversamente, il calcolatore genera dei segnali elettrici che fanno sì che la telescrivente stampi l'informazione richiesta. Un altro dispositivo di uscita è la *stampante ad alta velocità*, che può stampare assai velocemente molte linee di informazione.

Dati e programmi possono anche essere perforati su schede. Queste vengono lette da un *lettore di schede*, che rileva i fori delle schede e in base ad essi genera gli appropriati segnali elettrici per il calcolatore. La perforazione delle schede è fatta tramite un apposito dispositivo munito di tastiera, chiamato *perforatore di schede*.

Esistono anche altri dispositivi di ingresso/uscita. Noi ci siamo limitati a parlare solo di alcuni fra i più comuni.

Un calcolatore è fatto in modo da interpretare una serie basilare di comandi, detta *linguaggio macchina*. Un tipico comando di linguaggio macchina è ad esempio quello che realizza la memorizzazione dei numeri nella memoria principale; un altro è

quello che somma due numeri memorizzati in due differenti parti della memoria principale, quindi memorizza la somma in un'altra parte della memoria.

Scrivere programmi in linguaggio macchina è estremamente noioso, ed appunto per venire incontro ai programmatori, si sono sviluppati degli speciali linguaggi di programmazione. I programmi scritti in questi linguaggi in genere richiamano le normali espressioni. Supponiamo ad esempio di voler scrivere un programma che sommi tre numeri, per ottenerne un quarto, cioè di calcolare

$$s = x + y + z \tag{1-1}$$

dove x, y e z sono tre numeri, ed s è la loro somma. In linguaggio macchina, sarebbero necessari diversi passaggi; usando il linguaggio di programmazione BASIC, basta una sola istruzione:

```
10   LET S = X + Y + Z                                        (1-2)
```

Come si può vedere, l'istruzione BASIC è simile all'espressione algebrica. Vedremo presto che è sufficiente conoscere poche regole per scrivere senza difficoltà dei semplici programmi in BASIC.

In realtà sono i programmi scritti in linguaggio macchina a determinare tutte le operazioni di un calcolatore. A tal fine esistono due tipi di programmi speciali, detti *compilatori* ed *interpreti*, che hanno il compito di tradurre i programmi scritti in BASIC, o in altro linguaggio di programmazione, nel linguaggio macchina. Ad esempio, quando voi scrivete un programma in BASIC, il compilatore lo traduce automaticamente nel linguaggio macchina, senza che voi vi rendiate conto del procedimento. Spesso i compilatori presentano delle caratteristiche particolari, di aiuto al programmatore: ad esempio possono generare messaggi di errore, in base ai quali è possibile individuare gli errori del proprio programma. È questa la cosiddetta messa a punto di un programma o debugging (ossia eliminazione dei "bugs", o errori) di cui parleremo diffusamente nel paragrafo 1-3 e nel Capitolo 9.

Prima che un programma, scritto in BASIC o in un altro linguaggio di programmazione, possa essere eseguito, dev'essere fatta la compilazione, ovvero il programma dev'essere trasformato in un programma in linguaggio macchina: è questo programma in linguaggio macchina quello che sarà effettivamente eseguito dal calcolatore.

1-1-1. Elaborazione a divisione di tempo e a lotti

Esistono due metodi di elaborazione, detti *a divisione di tempo* (*timesharing*) e *a lotti* (*batch*). Nell'elaborazione a lotti, l'utilizzatore prepara il proprio programma scritto in BASIC su schede perforate, ottenendo così un pacco di schede. L'intero pacco (in cui la prima e l'ultima scheda sono particolari, in quanto contengono fra le

altre cose l'indentificazione del programmatore) è quindi dato al centro di calcolo. Il programma viene compilato ed eseguito, e i risultati sono ritornati al programmatore. Dato che il centro di calcolo riceve molti programmi, che vengono poi eseguiti uno alla volta, occorre spesso un giorno o anche più prima che il programmatore abbia i risultati. Comunque l'elaborazione a lotti offre un alto rendimento, perché permette di utilizzare il calcolatore ventiquattr'ore su ventiquattro.

Un altro tipo di elaborazione è il timesharing, meno efficiente del precedente, ma generalmente più vantaggioso per il programmatore. Questi infatti scrive il proprio programma mediante una telescrivente collegata al calcolatore. Si ha così il vantaggio di *tempi morti* brevi: infatti il programma può essere eseguito subito dopo essere stato battuto, e di conseguenza i risultati si avranno rapidamente. Inoltre, cosa ancora più importante, il programmatore viene rapidamente informato degli errori presenti nel programma, e può correggerli senza dover aspettare che il programma sia restituito dal centro di calcolo, come nel caso di elaborazione a lotti.

Il timesharing sarebbe decisamente antieconomico se il calcolatore venisse utilizzato da una sola persona. In realtà le prestazioni del calcolatore sono divise fra molti utilizzatori, sfruttando il fatto che la pura e semplice elaborazione avviene con estrema rapidità, mentre l'attività dell'operatore alla telescrivente è al confronto estremamente lenta: anche un dattilografo molto veloce è lentissimo se lo si paragona con la velocità del calcolatore. Allora, tra una battuta e l'altra dei tasti della telescrivente, il calcolatore resta praticamente inattivo: è in questa pausa che la sua capacità di elaborazione è assegnata ad altri programmatori. Si noti però che il calcolatore può essere condiviso con più utilizzatori anche in altri modi di elaborazione.

Il linguaggio di programmazione BASIC si adatta perfettamente al timesharing: il principiante troverà nel BASIC un linguaggio facile da imparare, e nel timesharing la possibilità di correggere rapidamente i suoi errori.

Parliamo ora del linguaggio di programmazione BASIC.

1-2. IL BASIC

In questo paragrafo definiremo gli elementi del inguaggio di programmazione BASIC e scriveremo un programma molto semplice, mentre nel prossimo scriveremo programmi più complessi.

Il linguaggio BASIC dipende dal compilatore o dall'interprete che viene usato. Generalmente il BASIC, nel suo complesso, è standard su tutti i calcolatori; tuttavia a volte si operano delle modifiche che ne aumentano la versatilità. In genere queste modifiche potenziano, lasciandone salva la validità, alcune delle operazioni di cui si parla in questo libro. In tutti i centri di calcolo esiste un manuale BASIC, che può essere consultato per determinare se ci siano modifiche di cui tener conto nel proprio programma.

1-2-1. I caratteri del BASIC

Tutti i linguaggi scritti, compresi i linguaggi di programmazione, consistono in una serie di caratteri. Il discorso vale anche per il BASIC. Il BASIC utilizza le 26 lettere dell'alfabeto, maiuscole:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

e le dieci cifre

0 1 2 3 4 5 6 7 8 9

Oltre ai numeri e alle lettere, si usano diversi simboli aritmetici e segni di punteggiatura:

+ − * / = ↑ < > < = < > = > ( ) , ; . : " # $ % & ? @

ed anche lo spazio di separazione. Vedremo più avanti come tutti questi caratteri sono impiegati nella stesura di un programma in BASIC.

1-2-2. La struttura del BASIC

Tutti i linguaggi hanno una struttura. In italiano, ad esempio, ogni frase deve avere un soggetto, un verbo, etc.; le frasi cominciano con una maiuscola e terminano con un punto, etc. Nei linguaggi di programmazione esistono regole analoghe, che devono essere rispettate esattamente. Vediamole.

Ogni linea in un programma BASIC si chiama *dichiarazione* (*statement* in inglese), ed è costituita da una sola istruzione. Ad esempio, consideriamo quanto segue:

```
10   REM PROGRAMMA CAMPIONE
20   LET A = 10
30   LET B = 25
40   LET C = A + B
50   PRINT C, A, B
60   END
```
(1-3)

Questo è un programma estremamente semplice, che realizza la somma di 10 e 25. Il risultato (35), così come 10 e 25, verranno stampati durante l'esecuzione del programma. Per il momento comunque lasciamo da parte l'operatività del programma, ed occupiamoci unicamente della struttura degli statements.

Ogni statement BASIC è costituito dal numero di statement, seguito da una *parola-chiave* BASIC, che indica il tipo d'istruzione. I numeri che individuano gli statements consistono in numeri interi privi di segno, di cinque cifre al massimo: in altre parole, non debbono avere il segno + o –, né il punto decimale. Quindi 1151, 10, 5500 sono numeri di statement validi, –10 e 11.1 non lo sono, il primo perché ha il segno, il secondo perché ha il punto decimale.

Le parole-chiave che compaiono nel programma (1-3) sono:

REM
LET
PRINT
END

Per ora soffermiamoci solo su queste; più avanti parleremo di altre parole-chiave.

REM sta ad indicare che lo statement è un commento (*remark* significa commento in inglese). Ciò vuol dire che in realtà il calcolatore, durante l'esecuzione del programma, ignora lo statement in questione. I commenti sono inseriti soltanto per rendere comprensibile il programma alla persona che lo legge.

LET fa sì che un valore venga assegnato ad una variabile: nel nostro caso, negli statements 20, 30 e 40 le variabili sono rispettivamente A, B e C. Le variabili rappresentano in realtà delle locazioni della memoria principale. Lo statement di LET è formato dalla parola LET seguita dalla variabile, dal segno uguale e, in generale, da un'espressione matematica. Quando il calcolatore esegue lo statement di LET, il risultato dell'operazione matematica è assegnato alla variabile. Allora, quando è eseguito lo statement 20 del nostro esempio, 10 viene memorizzato in una locazione di memoria indicata con A; analogamente, all'esecuzione dello statement 40, i numeri memorizzati nelle locazioni di memoria indicate con A e B vengono sommati, e la somma viene memorizzata in una locazione di memoria indicata con C.

PRINT fa sì che le quantità memorizzate nella locazione (o nelle locazioni) di memoria designata vengano stampate. Ad esempio, lo statement 50 del programma (1-3) fa si che i valori dei numeri memorizzati nelle locazioni di memoria C, A e B siano stampati sulla telescrivente dell'utilizzatore. Si noti che nello statement di PRINT fra le variabili sono inserite delle virgole: in seguito parleremo dettagliatamente di questa struttura.

END dev'essere l'ultimo statement di un programma. Svolge due funzioni: segnala al compilatore che il programma è terminato, e, quando il programma è eseguito, pone fine all'elaborazione.

Durante l'esecuzione di un programma in BASIC gli statements sono *eseguiti in conformità ai numeri di statement*, indipendentemente dall'ordine in cui sono scritti. Ad esempio, il programma che segue è identico al programma (1-3):

```
10  REM PROGRAMMA CAMPIONE
15  REM EQUIVALENTE AL PRECEDENTE
50  PRINT C, A, B
40  LET C = A + B
30  LET B = 25
20  LET A = 10
60  END                                        (1-4)
```

Si tenga presente che gli statements 20, 30 e 40 sono eseguiti in quest'ordine prima dello statement 50; in caso contrario il programma non potrebbe eseguire l'operazione richiesta perché C non può essere stampata prima di venir calcolata. Lo statement di END deve avere il numero di statement più alto di tutti.

Si tenga presente che i numeri che individuano gli statements in un programma non devono necessariamente differire di un'unità per volta. Nel caso del programma (1-3) si saltano 10 numeri fra uno statement e l'altro: questo è opportuno, di solito, perché permette all'utilizzatore d'inserire facilmente uno statement dimenticato fra due statements esistenti. Ad esempio, nel programma (1-3), volendo inserire uno statement dopo lo statement numero 40 e prima dello statement numero 50, si potrà farlo assegnandogli il numero 45: con questa numerazione il nuovo statement può essere battuto sulla telescrivente in qualsiasi momento.

Ed ecco ora alcune regole da seguire per scrivere un programma in BASIC, presupponendo di lavorare con un sistema in timesharing: in un paragrafo successivo tratteremo le modifiche che si attuano nel caso di elaborazione a lotti.

1) Gli statements devono essere scritti ciascuno su un'*unica* linea. Normalmente su una riga di telescrivente sono ammessi 72 caratteri, ma, a volte, *fino a* 132 caratteri. Per non aver dubbi su questo punto, è opportuno consultare il manuale BASIC accluso al calcolatore.

2) Tutti gli statements devono cominciare con un numero di statement, che deve a sua volta essere seguito da una parola-chiave BASIC.

3) Non si tiene conto degli spazi di separazione, necessari solo per rendere più leggibile lo statement a chi esamina il programma.

4) Gli statements si possono scrivere in un ordine qualunque, perché in ogni caso verranno eseguiti secondo l'ordine della numerazione. Lo statement di END deve avere il più alto numero di statement.

Si tenga presente che due statements non possono avere la medesima numerazione: in realtà, se vengono battuti più statements con lo stesso numero di statement, quello che è stato battuto per ultimo sostituisce tutti gli altri, e sarà l'unico ad essere eseguito dal calcolatore.

Un numero di statement seguito da linea bianca viene ignorato all'esecuzione del programma.

Dopo che è stata battuta una linea, occorre premere il tasto di ritorno carrello della telescrivente: questo indica che la linea è stata completata. Normalmente a questo punto, cioè quando è premuto il ritorno carrello, il calcolatore genera dei segnali che fanno sì che la carta inserita nella telescrivente avanzi di una linea. (Si tenga presente che la telescrivente ha un tasto, il ritorno carrello, che fa tornare il carrello al margine sinistro, ed un altro, l'avanzamento di linea, che fa avanzare la carta di una linea.) A volte vengono usati altri simboli, come appunto l'avanzamento di linea, per evidenziare che si è arrivati a fine riga. Consultando il manuale BASIC che correda il vostro calcolatore, vedrete quale simbolo dovete usare. (Se il tasto di ritorno carrello viene premuto *immediatamente dopo* il numero di statement, cioè senza avere poi aggiunto caratteri di spazio, la linea in questione sarà eliminata dal programma.)

Il linguaggio di programmazione BASIC è talmente comprensibile, che già con i primi rudimenti è possibile scrivere quasi subito dei semplici programmi. Nel prossimo paragrafo parleremo di come si scrivono programmi semplici.

1-3 ALCUNI SEMPLICI E COMPLETI PROGRAMMI IN BASIC

In questo paragrafo tratteremo della scrittura e dell'esecuzione di completi programmi in BASIC. Nel paragrafo 1-2 abbiamo già esaminato un semplice programma; ora svilupperemo il discorso facendo riferimento ai concetti già esposti.

Nell'esempio (1-3) abbiamo illustrato un semplice programma che sommava i numeri 10 e 25 ottenendo come risultato 35. Scriviamo ora questo programma in una forma diversa: si ricordi che tutti gli statements di un programma BASIC devono essere numerati ed avere una parola-chiave.

```
10  REM SEMPLICE PROGRAMMA DI
15  REM SOMMA DI DUE NUMERI
20  READ A, B
30  LET C = A + B
40  PRINT C, A, B
50  DATA 10, 25
99  END
```

(1-5)

Abbiamo introdotto qui due nuove parole-chiave del BASIC: READ e DATA. Esaminiamole. La parola-chiave READ instruisce il calcolatore a prelevare dei dati ed a memorizzarli in locazioni di memoria denominate appropriatamente. Nel nostro caso, lo statement 20 istruisce il calcolatore a memorizzare i dati nelle locazioni di memoria chiamate A e B. Si ricordi che è conveniente parlare di A e di B come di variabili. Si noti pure l'unica virgola che compare nello statement di READ: in uno statement di READ possono essere elencate molte variabili, ciascuna delle quali dev'essere separata dalla precedente mediante una virgola. Ad esempio, un altro statement di READ corretto sarà

```
50 READ A, B, C, D, F
```
(1-6)

I valori numerici da attribuire alle variabili sono forniti dallo statement di DATA. Allora, nel programma (1-5), all'esecuzione dello statement 20 il valore 10 sarà memorizzato nella locazione denominata A, mentre il valore 25 sarà memorizzato nella locazione denominata B. In altre parole, 10 sarà assegnato ad A e 25 a B.

Gli statements di READ e di DATA vengono usati insieme. Nel Capitolo 3 vedremo che è ammessa una notevole libertà nello scrivere gli statements di DATA. Comunque, per il momento, e per semplificare, saranno considerate soltanto alcune semplici regole. I dati contenuti nel programma (1-5) sono relativi allo statement di READ: cioè alla prima variabile presente nello statement di READ è associato il primo numero che compare nello statement di DATA, alla seconda variabile dello statement di READ è associato il secondo numero che compare nello statement di DATA. I numeri nello statement di DATA sono separati da virgole.

Gli statements di READ e di DATA non hanno necessariamente una corrispondenza uno ad uno. Illustriamo questo fatto con un programma che ha più di uno statement di READ e più di uno statement di DATA:

```
 10  REM STATEMENTS DI READ E DATA
 20  READ P, Q
 30  READ A
 40  READ D, M
 50  LET X = P + Q - A - D - M
 60  PRINT X
 70  DATA 21.6, 15.3, 22.4
 80  DATA 19.7
 90  DATA 84.3
100  END
```
(1-7)

Tutti gli statements di READ e di DATA hanno l'effetto di assegnare alle variabili P, Q, A, D, M i seguenti valori:

P = 21.6
Q = 15.3
A = 22.4
D = 19.7
M = 84.3 (1-8)

Chiariamo meglio il meccanismo. Tutti gli statements di DATA, considerati nel loro complesso, forniscono una lista di dati: questa lista è detta *blocco di dati*. I valori numerici dei dati sono posti nella lista nell'ordine in cui appaiono negli statements di DATA, presupponendo che questi siano ordinati secondo i rispettivi numeri di statement (cioè, lo statement di DATA con il numero più basso è scritto per primo, e così via). Per questo programma, abbiamo il seguente blocco di dati:

21.6
15.3
22.4
19.7
84.3 (1-9)

Supponiamo che vi sia un *puntatore* che punta ad un numero del blocco di dati. Alla partenza dell'esecuzione, il puntatore viene posizionato sul primo numero del blocco di dati. Alla partenza dell'esecuzione, il puntatore viene posizionato sul primo numero del blocco di dati. All'esecuzione di uno statement di READ, il numero individuato dal puntatore viene letto, e poi il puntatore è posizionato sul successivo numero del blocco di dati. In particolare, quando lo statement 20 del programma (1-7) è eseguito, a P è assegnato il valore del primo numero del blocco di dati, cioè 21.6. Il puntatore è ora posizionato sul secondo numero, per cui a Q sarà assegnato il valore 15.3. A questo punto il puntatore è avanzato sul terzo numero del blocco di dati, su cui rimane fermo finché non è eseguito lo statement 30, che assegna ad A il valore 22.4 e quindi posiziona il puntatore sul quarto numero del blocco di dati.

La procedura descritta è ripetuta finché tutti gli statements di READ non sono eseguiti.

In conclusione, i dati sono attribuiti alle variabili secondo la corrispondenza mostrata in (1-8).

Occorre notare che questa procedura di scrittura degli statements di READ e di DATA si rivela conveniente in molti casi. Supponiamo, sempre riferendoci al programma (1-7), che questo debba essere eseguito molte volte, con i valori di P, Q, D ed M mantenuti generalmente costanti, ed il valore di A che cambia da esecuzione ad esecuzione. Se noi scriviamo il valore numerico da assegnare ad A in uno state-

ment di DATA a parte, realizziamo facilmente il cambiamento desiderato. In generale, infatti, ogni statement di un programma in BASIC corrisponde ad una singola scheda perforata o ad una linea di telescrivente: allora, per cambiare il valore da attribuire ad A, è sufficiente cambiare solo una scheda perforata o ribattere una sola linea di testo, ed in tal modo è ridotto il lavoro di riedizione del programma.

Quando il programma è eseguito, la risposta (o le risposte) richiesta/e vengono stampate. Così, per il programma (1-5), verrà stampato il valore

35

mentre per il programma (1-7) verrà stampato il valore

−89.5

Il formato di stampa può, a volte, essere diverso da quello a cui siamo abituati. Supponiamo ad esempio che il risultato sia il numero 8,500,000,000.

Un modo usuale per rappresentare questo numero è 8.5×10^9, in quanto $10^9 =$ 1,000,000,000 (1 seguito da 9 zeri). Dato però che un terminale o una stampante non sono in grado di scrivere le soprascritte, 8.5×10^9 si scrive nella forma 8.5E9.

Secondo questa convenzione, E seguita da un numero rappresenta 10 elevato alla potenza di quel numero. Pertanto, le seguenti espressioni sono tutte equivalenti:

$$8{,}500{,}000{,}000 = 8.5 \times 10^9 = 8.5E9 = 8.5E+9$$
$$.00000012467 = .12467 \times 10^{-6} = .12467E-6$$

Si tenga presente che il segno più (+) che segue la E può essere omesso, e che il numero che segue la E deve essere intero (cioè non deve avere il punto decimale).

Questa notazione si usa quando sono stampati, come risposte, numeri di valore molto grande o molto piccolo, ed anche quando occorre fornire dati numerici al calcolatore.

1-3-1. Le variabili

Finora abbiamo supposto che le variabili siano rappresentate da lettere singole. In realtà in BASIC le variabili, che rappresentano numeri, possono essere lettere singole senza segno oppure lettere seguite da un intero. Ad esempio,

A1
B2
C
D9
E0

sono tutte variabili rappresentate correttamente; non così

AB
A.1
B–2

dal momento che non sono né lettere singole né lettere singole seguite da un intero: AB è formata da due lettere; A.1 ha un punto decimale; B–2 ha un segno.

In un successivo paragrafo parleremo delle variabili che possono rappresentare lettere, e delle relative regole di rappresentazione, parzialmente diverse da quelle ora esposte.

1-3-2. Gli operatori matematici

In BASIC, i simboli per l'addizione e la sottrazione sono rispettivamente:

+ e –

I simboli per le altre operazioni matematiche sono:

moltiplicazione	* (asterisco)
divisione	/ (sbarra)
esponenziale (elevamento a potenza)	↑ (freccia verticale)

Consideriamo, ad esempio, le formule algebriche

$$x = a / b \quad (1\text{-}10a)$$
$$y = c^2 \quad (1\text{-}10b)$$
$$z = (a + b)\ (c + d) \quad (1\text{-}10c)$$

In BASIC gli statements corrispondenti sono:

```
20   LET X = A/B            (1-11a)
40   LET Y = C↑2            (1-11b)
15   LET Z = (A + B)*(C + D)   (1-11c)
```

In BASIC le parentesi sono usate nello stesso modo che in algebra (v. (1-11c)), ma si tenga presente che fra le due parentesi della moltiplicazione *deve* comparire un asterisco. Ad esempio,

$$(A + B)(C + D)$$

è errato; in BASIC si deve scrivere:

$$(A + B) * (C + D)$$

Le parentesi possono essere usate per assicurare che le operazioni siano eseguite nel giusto ordine. Ad esempio,

$$A * B - C \qquad (1\text{-}12)$$

è equivalente ad $(a \cdot b) - c$, mentre

$$A * (B - C) \qquad (1\text{-}13)$$

è equivalente ad $a(b - c)$. Quindi le parentesi sono necessarie nel secondo caso. Nel paragrafo 2-3 discuteremo questo punto con maggiori dettagli. Per il momento, diciamo che, per assicurare l'esecuzione delle operazioni aritmetiche nell'ordine corretto, o utilizziamo le parentesi, o spezziamo l'operazione aritmetica su più statements. Supponendo ad esempio di voler calcolare $a(b - c)$, o usiamo la (1-13), o scriviamo i due statements

```
15  LET X1 = B - C
20  LET X2 = X1*A
```

Come ulteriore esempio di programma in BASIC, scriviamo un programma che calcola le radici dell'equazione di secondo grado

$$ax^2 + bx + c = 0 \qquad (1\text{-}14)$$

Le radici di questa equazione sono:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad (1\text{-}15a)$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \qquad (1\text{-}15b)$$

Supponiamo che $b^2 - 4ac$ sia positivo o zero (nel paragrafo 4-4 vedremo come questa restrizione può essere eliminata). Il programma è il seguente:

```
10  REM PROGRAMMA PER CALCOLARE LE RADICI
15  REM DI UNA EQUAZIONE QUADRATICA
20  READ A,B,C
30  LET D = (B↑2 - 4*A*C)↑.5
40  LET X1 = (-B + D)/(2*A)
50  LET X2 = (- B - D)/(2*A)
60  PRINT X1,X2
70  DATA 1,16,3
80  END                                        (1-16)
```

Esaminiamolo in dettaglio. Gli statements 10 e 15 sono un commento che viene ignorato quando il programma è eseguito.

Lo statement 20 fa sì che i valori di A, B, C siano letti: a causa dello statement di DATA (70) si ha: A = 1, B = 16, C = 3.

Il valore di $b^2 - 4ac$ è calcolato nello statement 30. Nel Capitolo 2 tratteremo l'ordine in cui sono eseguite le operazioni matematiche; per il momento, e per qualunque dubbio, utilizziamo le parentesi.

Negli statements 40 e 50 sono calcolati i valori di x_1 e x_2. Si tenga presente che, per rendere negativa una variabile, la si può far precedere dal segno meno; pertanto le espressioni

```
-X*2   e   (-1)*X*2
```

sono equivalenti: infatti entrambe, per X = 8, danno come risultato −16. Due simboli di operatori non possono comparire l'uno dopo l'altro. Così l'espressione A*−B è errata; l'espressione −A*B è invece lecita.

Lo statement 60 stampa i valori di X1 e X2. Ad esempio, con i dati forniti, sarà stampato come risultato

```
-.1897503     -15.81025
```

1-3-3. Ulteriori osservazioni sui commenti

I commenti possono essere introdotti in qualunque punto del programma. Con riferimento al programma (1-16), supponiamo di voler commentare con la frase "calcola la radice quadrata" lo statement 30: potremo far precedere lo statement 30 dal seguente:

```
25  REM CALCOLA LA RADICE QUADRATA
```

C'è comunque un altro modo, disponibile in alcune versioni di BASIC: il commento può essere inserito sulla *stessa* linea dello statement che si vuole commentare, preceduto da un apice. Quando in uno statement compare un apice, esso, e tutto ciò che lo segue, sarà ignorato durante l'esecuzione del programma. Pertanto, lo statement 30 del programma (1-16) potrebbe essere scritto nel modo seguente:

```
30   LET D = (B↑2 − 4*A*C)↑.5 'CALCOLA LA RADICE QUADRATA
```

In fase di esecuzione del programma, questo statement è esattamente equivalente allo statement 30 del programma (1-16). L'introduzione di commenti in fase di scrittura del programma aiuta un qualunque programmatore nella lettura del programma stesso.

1-3-4. Messa a punto dei programmi

Quando un programma è scritto, spesso contiene degli errori, detti in inglese *bugs*. La correzione di questi errori è chiamata messa a punto o *debugging*. Nel Capitolo 9 analizzeremo in dettaglio questo aspetto; qui accenneremo soltanto ad alcune semplici procedure di messa a punto.

Il tipo più semplice di errore è l'*errore di stampa*. Ad esempio, supponiamo che la linea 20 del programma (1-16) sia stata scritta in questo modo:

```
20   REND A,B,C
```

La parola READ è stata battuta in modo inesatto. In fase di compilazione del programma, sarà stampato un messaggio di errore, che segnalerà appunto la presenza di un errore nello statement 20. Una volta che sia stato segnalato un errore in uno statement, lo si può individuare rivedendo attentamente lo statement in questione. Allora, nel caso di elaborazione a lotti, si sostituisce la scheda che contiene l'errore con una corretta; nel caso del timesharing, si ribatte semplicemente lo statement errato. Nel paragrafo 1-4 tratteremo questo punto.

Esistono poi altri tipi di errori, chiamati errori *logici*: in questo caso il programma è ineccepibile dal punto di vista della scrittura in BASIC, ma il valore calcolato è errato. Ad esempio, se nel programma (1-16) la linea 50 è scritta come

```
50   LET X2 = (−B − D)/3*A
```

benché lo statement sia corretto, la risposta sarà errata, perché abbiamo diviso l'espressione per 3*A invece che per 2*A. Per individuare errori di questo genere, occorre che i risultati forniti dal programma siano ottenuti da dati d'ingresso per i quali,

per altra via, conosciamo la risposta. Ad esempio, per verificare la (1-16), dovremmo calcolare le radici di $x^2 + 16x + 3$ con una calcolatrice tascabile, e quindi confrontare il risultato con quello fornito dal calcolatore: se i risultati non coincidono, dobbiamo esaminare attentamente il programma, leggendo, ad esempio, ogni statement, ed esaminandone accuratamente gli effetti.

Solo dopo essersi accertati che il programma è corretto, esso può essere utilizzato senza più verificare l'esattezza del risultato. (Da notare che non ha senso scrivere il programma (1-16) per utilizzarlo *una sola volta*: la verifica del risultato tramite la calcolatrice tascabile vi ha risolto il problema per questo unico caso! Comunque, una volta che il programma è stato corretto, può essere rieseguito molte volte con dati diversi.)

In seguito analizzeremo altre procedure di messa a punto. È da notare che spesso è più facile verificare se il risultato è corretto che calcolare il risultato. Ad esempio, è più facile verificare se un numero è la radice di un'equazione di quinto grado piuttosto che risolvere l'equazione: in questo caso non è necessario ricorrere ad una calcolatrice, fermo restando che i risultati debbono essere sempre verificati.

Finora abbiamo scritto (e corretto) dei semplici programmi in BASIC. In realtà le informazioni fornite sono sufficienti per scrivere molti semplici programmi in BASIC. Prima di andare avanti, analizzeremo le procedure necessarie per l'esecuzione di programmi a lotti e in ambiente timesharing.

1-4. ESECUZIONE DI PROGRAMMI BASIC IN TIMESHARING

Finora abbiamo esaminato la scrittura di semplici programmi in BASIC. In questo paragrafo esamineremo come i programmi saranno eseguiti su un calcolatore in timesharing: più precisamente, descriveremo in qual modo si operi per introdurre ed eseguire un programma da terminale. Il BASIC si adatta perfettamente all'elaborazione in timesharing, ed è comunque adatto anche all'elaborazione a lotti (trattata nel paragrafo seguente). In ogni modo consigliamo la lettura di questo paragrafo anche a chi lavori a lotti, perché il discorso sarà comunque utile.

Le tecniche per introdurre ed eseguire un programma non fanno parte del linguaggio BASIC; pertanto le procedure che esporremo non sono standard, differendo da calcolatore a calcolatore. Consultando il manuale di BASIC che correda il calcolatore utilizzato potrete conoscere i particolari del caso. Qui saranno descritte le procedure tipo: in generale i concetti di base saranno i medesimi per tutti i calcolatori, anche se ci potranno essere delle differenze specifiche.

1-4-1. Inizio elaborazione (logging in)

Innanzitutto il vostro terminale dev'essere collegato al calcolatore. Normalmente il collegamento è fatto tramite linea telefonica, mediante uno speciale accoppiatore.

Supposto che tutti gli opportuni collegamenti siano realizzati, componiamo il numero telefonico del calcolatore. Una risposta del tipo

```
ABC COMPUTER CENTER
PLEASE LOG IN
```

apparirà sul vostro terminale. Per comunicare effettivamente col calcolatore, occorre farsi identificare. Qui indicheremo una sequenza che serve a questo scopo. Può darsi che il vostro calcolatore ne usi una diversa, ma i concetti di base sono i medesimi. Il calcolatore stampa un simbolo per indicare che è in attesa di una risposta: in questo caso abbiamo utilizzato come simbolo il punto. Allora una sequenza di log tipica è:

```
ABC COMPUTER CENTER
PLEASE LOG IN
.LOG
USER NUMBER PLEASE
.1115
PASSWORD
.XWRD (Nota che XWRD non è stampato)
USER 1115 LOGGED IN 25-FEB-77 12:01                    (1-17)
```

Esaminiamo la sequenza, tenendo presente che le parole scritte dopo il punto sono scritte da *voi*, le altre dal *calcolatore*. Dopo che voi avete scritto LOG, il calcolatore richiede che vi identifichiate tramite un numero, e inoltre, per essere sicuro che si tratti di un utente autorizzato, e non già di uno che ha dato un numero tirando a indovinare, richiede anche una parola d'ordine. In questo caso, la parola d'ordine è XWRD. Di solito, quando l'utente batte la parola d'ordine sulla tastiera del terminale, essa non viene stampata, allo scopo di evitare che venga letta da chiunque maneggi la carta su cui è stampato il testo. (Alcuni calcolatori ribattono più volte sulla parola d'ordine caratteri generici allo scopo di renderla irriconoscibile.) Dopo che la corretta parola d'ordine è stata scritta, il calcolatore segnala di aver registrato l'utilizzatore: il punto finale indica che il calcolatore è pronto a ricevere i comandi.

1-4-2. Esecuzione di un programma BASIC

Il calcolatore è a questo punto in uno stato detto *modo monitor*. Per poter introdurre, e quindi eseguire un programma in BASIC, il calcolatore dev'essere posto in una modalità particolare, che chiameremo modo BASIC: per far questo, occorre fornirgli il comando opportuno. Consideriamo la sequenza

```
.CALL BASIC
OLD OR NEW --> NEW
NEW FILE NAME --> SOMMA
>                                                  (1-18)
```

Dopo il punto, occorre scrivere CALL BASIC. Questo indica al calcolatore che si desidera entrare in modo BASIC. Ora si possono introdurre dei programmi in BASIC e/o eseguirli. Il calcolatore a sua volta risponde con OLD OR NEW -->. Se si desidera introdurre un programma nuovo, si batte NEW; se invece si vuol far eseguire un programma che è già stato memorizzato, si batte OLD. (Più avanti, in questo stesso paragrafo, approfondiremo questo punto.) Supponiamo per ora che si voglia introdurre un nuovo programma. Si batte dunque NEW; allora il calcolatore risponde con NEW FILE NAME -->. Dopo i due trattini ed il simbolo >, si batte il *nome* del programma che si desidera scrivere. Il nome di un programma è costituito da un massimo di 6 lettere o numeri. Il primo carattere *deve* essere una lettera. Si tenga presente che un *archivio* (*file* in inglese) può essere costituito da qualsiasi informazione si desideri memorizzare. Nel caso in questione, l'archivio è il programma. (Nel Capitolo 11 esamineremo altri tipi di archivio.) In questo esempio, al programma daremo il nome di SOMMA. Il calcolatore allora batterà il simbolo > sulla linea successiva. Il simbolo > è equivalente al punto nel modo monitor, e sollecita l'operatore segnalando che il calcolatore è pronto a ricevere altre informazioni.

La sequenza mostrata come esempio è tipica, ma può variare da calcolatore a calcolatore. In alcuni calcolatori, ad esempio, la sequenza d'ingresso potrà essere la seguente:

```
.CALL BASIC
READY
NEW (battuto dal programmatore)
NEW FILE NAME --- SOMMA
READY                                              (1-19)
```

Si osservi che dopo il primo READY bisogna battere OLD o NEW senza attendere il carattere di sollecitazione: infatti possono non esserci segni come > a dirvi di cominciare a scrivere il programma, bastando per questo la parola READY. Si possono avere anche altre varianti: ad esempio, CALL BASIC può essere sostituito da un'altra forma. Il manuale BASIC del calcolatore vi indicherà la forma corretta.

Illustriamo ora come si possa introdurre ed eseguire il programma (1-5). Diamo la lista della procedura completa, con esclusione della fase di log in:

```
.CALL BASIC
OLD OR NEW --> NEW
NEW FILE NAME --> SOMMA
```

```
>10  REM PROGRAMMA CAMPIONE DI SOMMA DI DUE NUMERI
>20  READ A,B
>30  LET C = A + B
>40  PRINT C,B,A
>50  DATA 10,25
>99  END
>RUN                                                (1-20)

SOMMA     11:14     17-FEB-77

35
>
```

Delle prime tre linee abbiamo già parlato (v. (1-18)). Il programma in BASIC introdotto è nelle sei linee seguenti. Si osservi che il calcolatore scrive il simbolo di sollecitazione (*prompt* in inglese) all'inizio di tutte le linee per le quali è richiesta la battitura di dati da parte del programmatore.

Dopo che il programma è introdotto, occorre battere RUN: il resto è scritto dal calcolatore. Più esattamente, nella prima linea compaiono il nome del programma, l'ora e la data; poi è scritto il risultato (o i risultati); infine, il calcolatore stampa >, per segnalare che è pronto a ricevere altre informazioni.

Anche qui sono possibili forme diverse: ad esempio, al posto del segno >, può essere battuta la parola READY. Si tenga presente che, quando si usa READY, questa non sostituisce ad uno ad uno i segni di sollecitazione: cioè (v. (1-19)), dopo che è stato stampato READY, è possibile introdurre il programma senza la comparsa del carattere di sollecitazione. Dopo aver battuto l'intero programma, occorre battere RUN, anche stavolta senza nessuna indicazione: il programma sarà eseguito come in (1-20). Al termine sarà stampata di nuovo la parola READY.

In questo libro saranno scritti soltanto programmi in BASIC, cioè programmi formalmente simili a (1-5); saranno omessi i dettagli riguardanti la denominazione del programma ed il modo in cui il programma viene introdotto ed eseguito.

Se nel programma ci sono errori, essi vengono facilmente identificati. Se ad esempio lo statement 30 del programma (1-20) fosse stato scritto come

```
30 LET C = A +
```

al RUN il calcolatore risponderebbe con

```
?ILLEGAL FORMULA IN LINE 30
```

L'errore così individuato può essere corretto senza difficoltà. Si noti il punto interrogativo che precede il messaggio di errore: esso sta ad indicare che il programma non può essere compilato correttamente. L'errore dev'essere corretto, prima che il

programma possa essere eseguito: per cambiare una linea, basta sovrapporgliene un'altra. In questo caso, ad esempio, si batterà:

```
>30   LET C = A + B
```

Questa procedura è utilizzabile anche per cambiare i dati. Supponaimo ad esempio che il programma sia stato eseguito correttamente con i dati forniti in (1-10). Dopo l'esecuzione del programma, potremmo scrivere:

```
>50   DATA 20,30
>RUN
```

In tal caso, con i nuovi dati, si avrebbe come risultato 50.

Si possono avere messaggi di errore anche per cause che intervengono durante l'esecuzione del programma. Ad esempio, può capitare che i numeri diventino troppo grandi perché il calcolatore li possa trattare, oppure che si abbia a che fare con la radice quadrata di un numero negativo. Si parlerà degli errori di questo tipo nel Capitolo 2.

1-4-3. Fine elaborazione (logging off) e salvataggio dei programmi

Quando si è concluso il lavoro con il calcolatore e si vuol terminare l'attività, si batte

```
>BYE
```

Il calcolatore stamperà un messaggio di fine elaborazione, insieme con la quantità di tempo di calcolatore impiegata ed altro. Generalmente il programma su cui si è lavorato andrà perduto. Volendo salvarlo per usarlo ancora in futuro, si batte

```
>SAVE
```

prima di eseguire la procedura di fine elaborazione: a questo punto, battendo BYE, il programma è salvato. Se in seguito si effettua la procedura di inizio elaborazione (logging in) e quindi si attiva il modo BASIC, si può tornare ad eseguire il programma. Per far questo, quando il calcolatore stampa OLD OR NEW ——>, occorre rispondere con OLD. La procedura è illustrata qui di seguito:

```
OLD OR NEW ——> OLD
OLD FILE NAME ——> SOMMA
>
```

A questo punto si può eseguire il programma SOMMA o si possono cambiare i dati esattamente come se non fosse mai stata eseguita la procedura di fine elaborazione (logging off). Non va dimenticato che i centri di calcolo non conservano gli archivi all'infinito: alcuni "cancellano" gli archivi non usati recentemente, per cui conviene controllare per quanto tempo i vostri programmi saranno conservati. Un altro punto da tener presente: se avete eseguito un programma e ora desiderate lavorare su un altro programma, non avete che da battere OLD o NEW in risposta al carattere di sollecitazione.

1-4-4. Comandi di sistema nel BASIC

Esistono diversi comandi di sistema che si usano per scrivere o eseguire programmi in BASIC. Noi ne abbiamo considerati alcuni: ad esempio, RUN fa eseguire il programma; NEW specifica che è stato creato un nuovo archivio (per il programma); di SAVE e OLD abbiamo già parlato. Vediamone ora qualcun altro.

UNSAVE è l'opposto di SAVE: se il programma con cui si è lavorato era stato precedentemente salvato, battendo UNSAVE al momento di eseguire la procedura di fine elaborazione l'archivio (programma) non sarà ulteriormente salvato.

SYSTEM fa uscire dal modo BASIC per ritornare al modo monitor, in modo che il calcolatore possa venire usato per altri tipi di operazioni. Si noti che tutti gli archivi che non risultano salvati a questo punto vanno perduti. Questo comando può variare a seconda dei calcolatori: a volte si usano MONITOR o MON.

GOODBYE è lo stesso di BYE.

LIST fa sì che venga stampato l'archivio corrente, cosa molto utile quando si vuol esaminare attentamente il programma su cui si sta lavorando.

CATALOG elenca i nomi di tutti gli archivi salvati.

SCRATCH cancella l'archivio corrente senza uscire dalla memoria. Se non si intende più riutilizzare un programma, è opportuno rimuoverlo dalla memoria, per evitare che questa sia occupata inutilmente. Nota: con alcuni calcolatori SCRATCH può determinare la cancellazione di *tutti* gli archivi. Pertanto, prima di utilizzare questo comando, occorre consultare attentamente il manuale del BASIC del calcolatore.

RENAME permette il cambiamento del nome dell'archivio su cui si lavora. Se ad esempio il programma su cui si lavora si chiama SOMMA, e si vuole cambiarne il nome in ADD, occorre scrivere

```
>RENAME ADD
```

A questo punto disponete delle cognizioni necessarie per far eseguire un programma in BASIC in ambiente timesharing. Chi non intende utilizzare l'elaborazione a lotti (batch processing) può saltare il prossimo paragrafo.

1-5. ESECUZIONE A LOTTI DI PROGRAMMI BASIC

In questo paragrafo esamineremo l'esecuzione a lotti di un programma in BASIC. Partiamo dal presupposto che si sia letto il paragrafo precedente sul timesharing, in quanto alcuni dei concetti là esposti sono applicabili anche qui.

Quando un programma in BASIC è elaborato a lotti, occorre perforare un pacco di schede. La porzione principale del pacco è occupata dal programma, tipo quello di (1-5), e ciascuno statement è contenuto in una scheda distinta. Il programma dev'essere preceduto e seguito da altre schede, che hanno il compito di controllare l'elaborazione: queste schede sono dette schede *di controllo del lavoro* (*job control* in inglese). Esaminiamole.

La prima scheda del pacco è detta *scheda di lavoro* (*job card* in inglese); essa segnala che un nuovo lavoro deve partire, e a volte riporta anche il numero e la parola d'ordine, in modo che il centro di calcolo possa registrare l'utilizzatore relativo. Questa scheda informa anche che il programma che segue va eseguito in BASIC. In realtà, tutte queste informazioni possono essere ripartite in più schede; cioè, ci può essere una sequenza di schede di controllo che precede il programma. Consultando il manuale del calcolatore potrete vedere qual è l'esatta sequenza delle schede di controllo. Noi comunque designeremo la sequenza suddetta con la singola scheda JOBCARD.

Il programma, una volta introdotto nel calcolatore, viene compilato. A questo punto dev'essere eseguito: questo è indicato da un'altra scheda, che chiameremo JOB EXECUTE. Anche il formato esatto di questa scheda andrà determinato in relazione al calcolatore utilizzato. In alcuni casi occorre anche una scheda per indicare che il lavoro è terminato: questa scheda la designeremo con JOBEND.

Per l'esatto formato, si consulti il manuale.

Volendo eseguire il programma (1-3) utilizzando l'elaborazione a lotti, l'intero pacco di schede sarà:

```
JOBCARD
10  REM PROGRAMMA CAMPIONE
20  LET A = 10
30  LET B = 25
40  LET C = A + B
50  PRINT C,B,A
60  END
JOBEXECUTE
JOBEND                                              (1-21)
```

Il pacco di schede dovrà essere consegnato al centro di calcolo. Quando il programma è eseguito, i risultati saranno stampati e voi potrete riceverli più tardi. Il formato della stampa del calcolatore sarà essenzialmente lo stesso di quello illustrato nell'ultimo paragrafo.

Se il programma, per la presenza di errori, non può essere compilato, non si avrà alcun risultato, ma sarà stampato un messaggio di errore, il cui formato ripete sostanzialmente quello visto nel paragrafo precedente.

A volte è necessario eseguire più volte uno stesso programma con differenti serie di DATA. I nuovi dati possono essere posti al termine del programma, preceduti da un'opportuna scheda di controllo. Come al solito, consultate il manuale del BASIC del calcolatore per definire l'esatto formato di tutte le schede di controllo e la posizione che dovranno avere all'interno del pacco di schede.

ESERCIZI

1-1. Descrivete le parti che compongono un moderno calcolatore numerico.

1-2. Qual è la differenza fra la memoria principale e la memoria su nastro magnetico?

1-3. Specificate i vari tipi di memoria ausiliaria del vostro calcolatore.

1-4. Qual è la funzione del compilatore?

1-5. Parlate dell'elaborazione a divisione di tempo (timesharing) e dell'elaborazione a lotti (batch processing).

1-6. Fate un elenco di tutti i caratteri usati nel linguaggio di programmazione BASIC.

1-7. Descrivete la struttura di uno statement BASIC.

1-8. Qual è la funzione del numero di statement?

1-9. Qual è la funzione della parola-chiave?

1-10. Illustrate la funzione della parola-chiave REM. Perché si usano i commenti?

1-11. Illustrate la funzione della parola-chiave LET.

1-12. Illustrate la funzione della parola-chiave PRINT.

1-13. Illustrate la funzione della parola-chiave END. Perché questa deve comparire nello statement che porta il numero più alto?

1-14. Illustrate la funzione delle parole-chiave READ e DATA. Perché si usano insieme?

1-15. Scrivete un programma in BASIC che calcoli

$$x = a - 3b + c$$

Eseguite questo programma sul vostro calcolatore con i seguenti valori:

$$a = 1.5 \quad b = 2.7 \quad c = 4$$

I valori di a, b, c ed x dovranno essere stampati.

1-16. Ripetete l'Esercizio 1-15 riferendovi alla funzione

$$x = \frac{(a + b)(c-b)}{a + c}$$

1-17. Ripetete l'Esercizio 1-15 riferendovi alla funzione

$$x = \frac{(a^2 + b^2)(c-a)}{c^2 + a^2}$$

1-18. Ripetete l'Esercizio 1-16, ma questa volta con i seguenti valori:

$$a = 2E5 \quad b = 3.17E4 \quad c = 16E4$$

1-19. Ripetete l'Esercizio 1-17, ma con i valori di a, b e c dati nell'Esercizio 1-18.

1-20. Ripetete l'Esercizio 1-15 riferendovi alla funzione

$$x = \frac{(a-b)^2(a + c)^{1/2}}{(a^2 + b^2)^{1/3}}$$

1-21. Scrivete un programma che calcoli le due funzioni

$$x = \frac{(a + b)(a^2 + c^2)^{1/2}}{a + b + c \cdot d}$$

$$y = \frac{(a + x)(d + x)^2}{(x^2 + d^2)^{1/5}}$$

in cui $a = 1.5$; $b = 2.37$; $c = 1.94$; $d = 6.21$. I valori di a, b, c, d, x ed y dovranno essere stampati.

1-22. Ripetete l'Esercizio 1-21 riferendovi alle funzioni

$$x = \frac{[(a^2-b^2)^2(a^2-d^2)^4]^{1/3}(a-b)}{(a^2-d^2)^2(a + b)}$$

$$y = \frac{[(x + a)^2(a-x)^4(x^2-d)^2 + (x + c)(x + b)]^{1/5}}{[(x + a)(x + b)(x + c)]^{1/3}}$$

1-23. Cercate gli errori presenti nel seguente programma in BASIC:

```
10  READ A.B
20  LET C = A - B
30  PRINT A,B.D
40  DATE 10,15
40  END
```

1-24. Il seguente programma in BASIC ricava la media di tre numeri. Cercate gli errori.

```
10,  READ A,B,C
20   LET D = A + B - C
30   PRINT D
40   DATA 10,15,-6
35   END
```

CAPITOLO 2

LE OPERAZIONI ARITMETICHE

In questo capitolo parleremo degli statements BASIC che si usano per eseguire le operazioni aritmetiche di addizione, sottrazione, moltiplicazione, divisione ed elevamento a potenza. Per maggior completezza, ripeteremo in parte quanto si è già detto nel Capitolo 1, ma molto più dettagliatamente. Useremo qui, come in tutto il libro, le lettere minuscole per rappresentare le espressioni algebriche, in modo da distinguerle dagli statements in BASIC, che utilizzano invece le maiuscole.

2-1. ADDIZIONE, SOTTRAZIONE, MOLTIPLICAZIONE E DIVISIONE

Consideriamo ora le espressioni BASIC usate per le operazioni aritmetiche di addizione, sottrazione, moltiplicazione e divisione.

2-1-1. Addizione e sottrazione

I simboli del BASIC per l'addizione e la sottrazione sono + e – rispettivamente. Consideriamo l'espressione algebrica

$$x = a + b + c + d \tag{2-1}$$

La corrispondente espressione in BASIC è:

```
50   LET X = A + B + C + D                                        (2-2)
```

Supponiamo che i valori di A, B, C e D siano già stati definiti nel corso del pro-

gramma, ad esempio nel modo che appare nel seguente segmento di programma (parte di un programma):

```
10   LET A = 20
20   LET B = 36.2
30   LET C = 41.7
40   LET D = 1
50   LET X = A + B + C + D                                    (2-3)
```

Si ricordi (v. par. 1-2) che, ogniqualvolta è fatta un'operazione di assegnazione relativamente ad una nuova variabile, un valore numerico è memorizzato in una locazione di memoria per quella variabile. Nel nostro esempio, lo statement 10 fa sì che il valore 20 sia memorizzato nella locazione di memoria detta A; lo statement 20 fa sì che il valore 36.2 sia memorizzato nella locazione di memoria detta B, e così via. Tutte le variabili che compaiono alla destra del segno uguale devono essere definite prima che lo statement BASIC relativo venga eseguito. Ad esempio, il seguente segmento di programma:

```
10   LET A = 20
20   LET B = 36.2
60   LET C = 41.7
40   LET D = 1
50   LET X = A + B + C + D                                    (2-4)
```

è inesatto, perché, dal momento che lo statement 60 è eseguito dopo lo statement 50, non si ha ancora il valore di C al momento dell'esecuzione dello statement 50 (sempre che C non sia già stata definita in un precedente statement).

Vediamo ora un esempio di sottrazione. Supponiamo di voler calcolare

$$y = a + b - c \qquad (2\text{-}5)$$

Useremo l'espressione BASIC seguente:

```
55   LET Y = A + B - C                                        (2-6)
```

Calcoliamo infine

$$f = a + b - c \qquad (2\text{-}7a)$$
$$g = f + a - b \qquad (2\text{-}7b)$$

dove a = 1.2; b = 2.6 e c = 0.7. Le due espressioni saranno calcolate dal seguente segmento di programma:

```
10   LET A = 1.2
20   LET B = 2.6
30   LET C = 0.7
40   LET F = A + B − C
50   LET G = F + A − B                                   (2-8)
```

Lo statement 40 deve precedere lo statement 50, perché, per poter calcolare G, deve essere già stato calcolato e memorizzato il valore di F.

Si badi che gli esempi precedenti sono segmenti di programmi, e non programmi completi. Quindi, volendo eseguire realmente il segmento (2-8), dovremo aggiungere gli statements di PRINT e di END, ad esempio nel modo seguente:

```
60   PRINT F,G
70   END                                                 (2-9)
```

2-1-2. Moltiplicazione e divisione

Come simbolo della moltiplicazione il BASIC usa l'asterisco *, perché i perforatori di schede non hanno il segno della moltiplicazione. Ad esempio, per calcolare l'operazione algebrica

$$c = ab \qquad (2\text{-}10)$$

scriveremo lo statement BASIC

```
20   LET C = A*B                                         (2-11)
```

Si noti che l'asterisco deve sempre comparire fra le quantità da moltiplicare: pertanto

```
20   LET C = AB
```

non è esatto.

Il simbolo del BASIC per la divisione è la sbarra /. Ad esempio, l'espressione algebrica

$$g = h/t = \frac{h}{t} = h \div t \qquad (2\text{-}12)$$

si scriverà in BASIC come segue:

```
30  LET G = H/T                                   (2-13)
```

Scriviamo a titolo di esempio un segmento di programma in BASIC che esegua le seguenti operazioni algebriche:

$$a = x + y$$
$$b = 2ax/g$$

dove $x = 1.3$; $y = 4.6$ e $g = 3.4$. Il segmento di programma sia:

```
10  LET X = 1.3
20  LET Y = 4.6
30  LET G = 3.4
40  LET A = X + Y
50  LET B = 2*A*X/G                               (2-14)
```

Si ricordi che, per fare un programma completo, bisogna aggiungere gli appropriati statements di PRINT e di END.

2-1-3. Il segno meno di negazione

Per rendere negativa una variabile, le si premette il segno meno. Questo segno è detto segno meno *di negazione*, perché non ha il compito di eseguire la differenza fra due quantità, ma agisce soltanto su una variabile. Il segno meno di negazione *non può* essere adiacente a nessun altro operatore matematico, quali +, *, /. Ad esempio, lo statement BASIC corrispondente a

$$c = -ab/b \qquad (2\text{-}15)$$

è

```
70  LET C = -A*B/D                                (2-16)
```

Invece lo statement seguente:

```
75  LET C = A*-B/D                                (2-17)
```

non è valido, perché il segno meno di negazione é adiacente al segno di moltiplicazione.

2-2. ELEVAMENTO A POTENZA

Tratteremo ora l'espressione BASIC usata per l'elevamento a potenza. Il simbolo usato è la freccia rivolta vero l'alto, ↑. Si ricorre a questo simbolo al posto dell'esponente perché i lettori di schede, le stampanti e le telescriventi non sono predisposti per introdurre o ricevere caratteri soprascritti. Quindi, lo statement BASIC corrispondente all'espressione algebrica seguente:

$$x = a^3 \qquad (2\text{-}18)$$

è

```
20   LET X = A↑3                                         (2-19)
```

A questo punto bisogna stare attenti. L'esponente può essere sia un numero intero che un numero decimale. Nel secondo caso, il numero elevato a potenza non dev'essere negativo. Ad esempio, il segmento di programma

```
10   LET A = 1.5
20   LET X = A↑3                                         (2-20)
```

è valido in BASIC. Come pure il seguente:

```
10   LET A = -1.5
20   LET X = A↑3                                         (2-21)
```

Si noti che il segmento (2-21) è valido anche se A è negativo, perché l'esponente è un numero intero. Invece il seguente segmento di programma, corrispondente all'espressione algebrica $x = a^{2.5}$,

```
10   LET A = -1.5
20   LET X = A↑2.5                                       (2-22)
```

non è valido, perché A è negativo e l'esponente *non* è un numero intero.

Gli esponenti non devono êssere necessariamente delle costanti, ma possono anche essere delle variabili. Consideriamo ad esempio le seguenti operazioni algebriche:

$$a = x + y$$
$$b = g + h - a$$
$$c = x + a/b$$
$$d = b^c \qquad (2\text{-}23)$$

dove x = 1.3; y = 2.6; g = 4.3 e h = 1.7. Un possibile segmento di programma BASIC corrispondente è:

```
10   LET X = 1.3
20   LET Y = 2.6
30   LET G = 4.3
40   LET H = 1.7
50   LET A = X + Y
60   LET B = G + H − A
70   LET C = X + A/B
80   LET D = B↑C                                   (2-24)
```

Si ricordi che tutte le variabili devono essere già state definite prima che possano comparire a destra del segno di uguale di un'espressione, e che tutte le quantità elevate ad un esponente devono essere positive. Nel paragrafo 4-4 analizzeremo le procedure da usare per affrontare problemi di questo tipo.

Se cercate di elevare un numero negativo ad una potenza non intera, il calcolatore stamperà un messaggio di errore. Altre versioni di BASIC operano la sostituzione del numero negativo con un numero positivo, e poi continuano il calcolo, ma anche in questo caso verrà stampato un messaggio di errore per avvertire della sostituzione, in quanto il risultato potrebbe essere per voi non significativo. Ad esempio, il segmento di programma (2-24) non presenterà problemi, dato che B è effettivamente positivo. Ma se cambiamo lo statement 10 in

```
10   LET X = 26.3                                  (2-25)
```

B diventa negativo, e allora durante l'esecuzione del programma verrà stampato il seguente messaggio:

```
% ABSOLUTE VALUE RAISED TO POWER IN LINE 50
```

Il programma comunque non s'interromperà. Se ad esempio rendiamo il segmento (2-24) un programma completo, aggiungendo

```
 90   PRINT D
100   END
```

e modifichiamo lo statement 10 come in (2-25), comparirà in uscita

```
% ABSOLUTE VALUE RAISED TO POWER IN LINE 50
1.11570E34
```

Si noti il segno % (per cento), che precede il messaggio di errore: in alcuni calcolatori questo segno sta ad indicare che l'errore è stato rilevato quando il programma era già in fase di esecuzione, e non in fase di compilazione. Invece i messaggi di errore preceduti dal punto interrogativo, ?, indicano che l'errore è stato rilevato in fase di compilazione. (Da un centro di calcolo all'altro possono esserci varianti.)

In generale, se si ha l'espressione BASIC

```
20   LET A = X↑Y
```

per X = 0, A sarà calcolato uguale a 0, mentre, per Y = 0, A sarà calcolato uguale ad 1. Invece occorre verificare direttamente qual è il valore di A nel caso che X ed Y siano entrambi uguali a zero, eseguendo un programma di prova.

2-2-1. Simboli alternativi per l'elevamento a potenza

Alcuni compilatori BASIC ammettono l'uso del doppio asterisco, **, per indicare l'elevamento a potenza. Ad esempio, lo statement BASIC

```
30   LET C = A↑B                                   (2-26a)
```

è equivalente al seguente:

```
30   LET C = A**B                                  (2-26b)
```

In generale, *tutte* le versioni di BASIC ammettono il segno ↑, mentre solo alcune ammettono *anche* il segno **. In questo libro, per indicare l'elevamento a potenza sarà usato sempre il segno ↑.

2-3. GERARCHIA DELLE OPERAZIONI ARITMETICHE

In BASIC le operazioni aritmetiche vengono eseguite in un certo ordine: si parla a tal fine di *gerarchia delle operazioni.* Del resto questa gerarchia esiste anche nell'aritmetica ordinaria. Vediamola. Supponiamo di avere l'equazione algebrica

$$x = a + bc + d/e \tag{2-27}$$

Sappiamo che questo vuol dire moltiplicare b per c, poi dividere d per e, e infine sommare a, bc e d/e. L'operazione può essere resa più evidente inserendo delle parentesi:

$$x = a + (bc) + (d/e) \tag{2-28}$$

In realtà le parentesi non sono effettivamente necessarie, dal momento che noi conosciamo l'ordine in cui devono essere eseguite le operazioni nell'algebra ordinaria.

Purtuttavia la gerarchia dell'algebra può essere modificata mediante parentesi. Supponiamo ad esempio di inserire delle parentesi nell'equazione (2-27) nel modo seguente:

$$x = (a + b)(c + d)/e \tag{2-29}$$

Questa volta il risultato è ottenuto come segue: sommiamo a e b, poi sommiamo b e c; quindi calcoliamo il prodotto di (a + b) per (c + d), e infine dividiamo il risultato per e.

Anche in BASIC esiste una gerarchia che può venir modificata per mezzo delle parentesi: per cominciare, comunque, esaminiamo le regole della gerarchia, e rimandiamo la discussione sull'uso delle parentesi al prossimo paragrafo.

Per esaminare la gerarchia, facciamo riferimento ad uno statement algebrico BASIC, ossia a quella parte dello statement che compare alla destra del segno uguale in uno statement di LET. Lo statement viene letto da sinistra a destra, cioè nel modo solito. Supponiamo che questa lettura venga fatta in tre tempi, come segue:

1) La prima volta sono eseguiti tutti gli elevamenti a potenza.

2) La seconda volta sono eseguite tutte le moltiplicazioni e/o le divisioni: tra moltiplicazioni e divisioni non vi è precedenza.

3) Infine vengono eseguite tutte le addizioni e/o le sottrazioni, ed anche in questo caso tra addizioni e sottrazioni non vi è precedenza. Il segno meno di negazione è considerato come un operatore di sottrazione. (Più avanti nel corso del paragrafo esamineremo questo punto, con eventuali modifiche.)

Possiamo dire che vi sono tre livelli gerarchici: il primo concerne l'elevamento a potenza; il secondo la moltiplicazione e la divisione; il terzo l'addizione e la sottrazione. Un dato statement può contenere più di un termine allo stesso livello di gerarchia: in tal caso essi vengono eseguiti nell'ordine, da sinistra a destra.

Illustriamo ora la gerarchia delle operazioni nel BASIC con qualche esempio. Consideriamo lo statement BASIC

```
20   LET X = A + B*C + D/E
```
(2-30)

nel quale manca l'elevamento a potenza. Allora, leggendo da sinistra a destra, eseguiremo prima B*C, poi D/E, e infine sommeremo A, B*C e D/E. Di conseguenza questo esempio corrisponde all'espressione algebrica (2-27).

Vediamo un altro esempio:

```
30   LET X = A + B*C↑3 + D/E↑.5
```
(2-31a)

Prima di tutto si esegue l'elevamento a potenza, per cui sono calcolati C↑3 ed E↑.5; poi B è moltiplicato per C↑3, e D è diviso per E↑5. Infine sono eseguite le addizioni. Pertanto l'equazione (2-31a) è equivalente all'espressione algebrica

$$x = a + b(c^3) + \frac{d}{e^{.5}} \qquad (2\text{-}31b)$$

Le parentesi non sono necessarie: le abbiamo inserite solo per chiarezza.

L'ultimo esempio è:

```
40   LET X = A/B*C - D*E↑F↑G
```
(2-32a)

Per primo si esegue l'elevamento a potenza, calcolando E↑F↑G. L'operazione è eseguita da sinistra a destra: quindi la prima operazione è E↑F, il cui risultato è poi elevato a G. Tutto ciò è equivalente all'espressione algebrica $(e^f)^g$. Successivamente si eseguono le moltiplicazioni e le divisioni, procedendo da sinistra a destra: dunque prima si calcola A/B, il cui risultato è quindi moltiplicato per C, poi D è moltiplicato per E↑F↑G. Infine si esegue la sottrazione. Pertanto la (2-32a) è equivalente all'espressione algebrica

$$x = \frac{a}{b}c - d(e^f)^g \qquad (2\text{-}32b)$$

Si tenga presente come sia importante l'ordine di lettura: A/B*C riconduce a a/b c, non ad a/(bc).

Le regole relative alla gerarchia, per quanto riguarda il segno *meno di negazione* e l'elevamento a potenza, seguono solitamente le regole ordinarie. Ovvero, –A↑B è equivalente a $-(a^b)$: cioè l'elevamento a potenza è eseguito prima della sottrazione. In qualche versione di BASIC, –A↑B è ricondotto a $(-a)^b$. Ciò accade raramente, ma bisogna verificarlo, o eseguendo un semplice programma di prova, o consultanto il manuale BASIC del calcolatore.

2-4. USO DELLE PARENTESI

In uno statement BASIC si possono inserire delle parentesi per modificare le regole di gerarchia. Le parentesi possono anche servire per essere sicuri dell'esito di una data operazione, quando non siamo sicuri della gerarchia.

Illustreremo come si usano le parentesi scrivendo alcuni semplici esempi; successivamente enunceremo le regole per il loro impiego. Consideriamo l'espressione algebrica

$$x = \frac{a + b + c}{d + e} \tag{2-33}$$

Se ci riferiamo solo alle regole di gerarchia, non possiamo scrivere in BASIC una singola espressione equivalente alla (2-33). Usando le parentesi nel modo solito, l'equazione (2-33) può essere scritta come segue:

$$x = (a + b + c)/(d + e) \tag{2-34a}$$

Allo stesso modo si possono inserire le parentesi in uno statement BASIC. Ad esempio,

```
26   LET X = (A + B + C)/(D + E)                    (2-34b)
```

è uno statement BASIC corretto, equivalente all'equazione (2-34a).

Le regole del BASIC relative alle parentesi sono sostanzialmente le stesse dell'algebra: le espressioni chiuse fra parentesi sono calcolate per prime, da sinistra verso destra. In altre parole, ciascuna espressione fra parentesi è considerata come un'espressione BASIC a parte, e viene calcolata secondo le regole di gerarchia. Eseguito il calcolo, ogni espressione fra parentesi è quindi trattata come se fosse una singola variabile. A questo punto l' "espressione BASIC" risultante è calcolata secondo le regole di gerarchia.

Illustreremo l'uso delle parentesi con vari esempi, il primo dei quali è il seguente:

```
35   LET X = (A + B)↑(C + D) – (E – F)*(G + H)      (2-35a)
```

I termini fra parentesi sono calcolati per primi: allora A è sommato a B, poi C è sommato a D, F è sottratto da E, e G è sommato ad H. A questo punto ogni espressione posta fra parentesi è considerata una variabile. Allora si esegue l'elevamento a potenza (A + B)↑(C + D), e successivamente la moltiplicazione (E – F)*(G + H). Infine si esegue la sottrazione. Pertanto la (2-35a) è equivalente a

$$x = (a+b)^{(c+d)} - (e-f)(g+h) \tag{2-35b}$$

Si tenga presente che in BASIC non sono ammesse le parentesi adiacenti ad indicare la moltiplicazione: (E − F) (G + H) *è errato.*
Bisognerà scrivere (E − F)*(G + H).
Consideriamo ora un secondo esempio:

```
20   LET X = (A+B*C)↑(C↑2+D)−(E+F*G)*(H+I/J)/K                    (2-36a)
```

Dapprima vengono calcolate le espressioni fra parentesi, secondo le regole di gerarchia, cioè, procedendo da sinistra a destra, le espressioni fra parentesi sono equivalenti, rispettivamente, ad $a + bc$, $c^2 + d$, $e + fg$, $h + i/j$. A questo punto le espressioni fra parentesi sono trattate come variabili, e l'espressione risultante è calcolata secondo le regole di gerarchia del BASIC. Lo statement BASIC scritto in (2-36a) è equivalente a

$$x = (a + bc)^{(c^2 + d)} - \frac{(e + fg)(h + i/j)}{k} \qquad (2\text{-}36b)$$

2-4-1. Annidamento di parentesi

A volte è opportuno, in uno statement BASIC, introdurre coppie di parentesi fra altre coppie di parentesi. Illustriamo questo punto; poi formuleremo le regole. Consideriamo l'espressione algebrica

$$x = [(a + b)/(c + d) + e][(a + g)^{(h-i)} + jk] \qquad (2\text{-}37a)$$

Abbiamo qui delle parentesi tonde entro parentesi quadre. In BASIC si hanno soltanto parentesi tonde, ma questo non limita la versatilità del BASIC, perché il BASIC contempla l'inserimento di parentesi tonde entro altre parentesi tonde. Ad esempio, uno statement BASIC equivalente all'espressione (2-37a) è

```
25   LET X = ((A+B)/(C+D)+E)*((A+G)↑(H−I)+J*K)                    (2-37b)
```

La forma dello statement (2-37b) corrisponde a quella dell'equazione (2-37a), con parentesi tonde al posto delle parentesi quadre. Per calcolare espressioni di questo tipo vale la regola che il contenuto delle parentesi più interne viene calcolato per primo, secondo le solite regole di gerarchia: queste parentesi interne possono essere quindi trattate come singole quantità. Allo stesso modo sono calcolate le parentesi più esterne; infine si calcola l'intera espressione. Illustreremo queste regole riferendoci all'espressione (2-37b). Dapprima sono calcolate le parentesi più interne (A+B), (C+D), (A+G) e (H−I). A questo punto, consideriamo ciascuna di que-

ste come una singola quantità, e calcoliamo le parentesi rimaste con le regole precedenti. Infine, si calcola l'intera espressione.

Come ulteriore esempio, si consideri lo statement BASIC

```
40   LET X = (A+(B-C*A)↑(A-C))/((A+C↑2)*(D-E*F↑G))        (2-38a)
```

Applicando le regole come si è già fatto nel precedente esempio, otteniamo l'espressione algebrica equivalente

$$x = \frac{a + (b-ca)^{(a-c)}}{(a + c)^2\,(d-ef^{g})} \qquad (2\text{-}38b)$$

Fin qui abbiamo esaminato il caso di coppie di parentesi annidate in un'altra coppia di parentesi. In realtà si può avere anche un grado più alto di annidamento. Ad esempio,

```
45   LET Y = ((A+B*(C-D↑2))*(E-F))/(G-H*(I+J))        (2-39a)
```

è equivalente all'espressione algebrica

$$y = \{[a + b(c-d^2)]\,(e-f)\} / [g-h(i + j)] =$$

$$= \frac{[a + b(c-d^2)]\,(e-f)}{g - h(i + j)} \qquad (2\text{-}39b)$$

Le regole seguite sono sempre le stesse. Dapprima si calcolano le coppie di parentesi più interne, che vengono quindi trattate come singole quantità. Poi si calcolano, e si trattano come singole quantità, le "rimanenti" parentesi più interne: la procedura viene ripetuta fino a calcolare tutte le parentesi. Queste vengono quindi trattate come singole quantità, ed infine si esegue il calcolo dell'intera espressione.

In genere il numero massimo di parentesi che possono essere annidate le une nelle altre è limitato: il manuale BASIC del calcolatore vi dirà il grado di annidamento consentito.

Come ultimo esempio, useremo le parentesi per ridurre il numero di statements BASIC nel programma (1-16), che ricava le radici di un'equazione di secondo grado. Consideriamo due statements che compaiono in quel programma:

```
30   LET D = (B↑2-4*A*C)↑.5
40   LET X1 = (-B+D)/(2*A)
```

Questi statements possono essere sostituiti da un unico statement:

```
30   LET X1 = (−B+(B↑2−4*A*C)↑.5)/(2*A)
```

Quindi possiamo scrivere il programma (1-16) come segue:

```
10   REM PROGRAMMA PER CALCOLARE LE RADICI
15   REM DI UNA EQUAZIONE QUADRATICA
20   READ A, B, C
30   LET X1 = (−B+(B↑2−4*A*C)↑.5)/(2*A)
40   LET X2 = (−B−(B↑2−4*A*C)↑.5/(2*A)
50   PRINT X1,X2
60   DATA 1,16,3
70   END                                         (2-40)
```

Si tenga presente che abbiamo mostrato come l'uso delle parentesi può ridurre il numero degli statements necessari; ma questo non è sempre un vantaggio. Ad esempio, nell'esecuzione del programma (2-40), B↑2−4*A*C verrà calcolato due volte: una prima volta all'esecuzione della linea 30, e una seconda volta all'esecuzione della linea 40. (Non bisogna dimenticare che il calcolatore segue esattamente le istruzioni.) Invece, nel programma (1-16), B↑2−4*A*C viene calcolato una sola volta; quindi il programma è eseguito in minor tempo. Nel caso in questione, la differenza è trascurabile, ma, nel caso di programmi complessi, un tale risparmio di tempo può essere importante.

2-4-2. Uguaglianza del numero delle parentesi aperte e chiuse

Le parentesi si usano a coppie, quindi, ogni volta che si scrive una parentesi sinistra, (, in un punto successivo dello statement la parentesi dev'essere chiusa con una parentesi destra ,). Per questa ragione ogni statement in BASIC deve avere un numero uguale di parentesi sinistre e parentesi destre. Scrivendo uno statement lungo, è facile dimenticare di pareggiare le parentesi: dunque, scrivendo uno statement in BASIC che contenga delle parentesi, contate le parentesi sinistre e destre per accertarvi che siano in numero uguale. Se il numero delle parentesi sinistre e destre in uno statement non è uguale, in fase di compilazione sarà inviato un messaggio di errore, e il programma non sarà eseguito.

2-4-3. Uso delle parentesi con il segno meno di negazione

Nel paragrafo 2-1 abbiamo trattato l'uso del segno meno di negazione; in particolare si è detto che espressioni del tipo A*−B sono errate. L'uso di parentesi ci per-

mette di scrivere statements equivalenti. Ad esempio, il seguente è uno statement ammesso dal BASIC:

```
25   LET A = X*(-B)+C                                        (2-41)
```

2-5. ASSEGNAZIONE DI VALORI. LET ED IL SEGNO DI UGUALE

Quando è eseguito uno statement di LET, il segno uguale ha un significato diverso da quello ordinario in algebra. Consideriamo il segmento di programma

```
10   LET A = 10.3
20   LET B = 5.6
30   LET C = A+B                                             (2-42)
```

Lo statement 10 fa sì che il valore 10.3 sia assegnato ad A, il che vuol dire che il valore 10.3 viene memorizzato in una cella di memoria che è denominata A. Allo stesso modo, lo statement 20 fa sì che 5.6 sia memorizzato in una cella di memoria denominata B. Lo statement 30 fa sì che i valori memorizzati nelle celle di memoria denominate A e B vengano sommati, e che la loro somma sia memorizzata in una cella di memoria denominata C. Si noti che i valori memorizzati nelle celle di memoria A e B non cambiano a causa di questo procedimento, di modo che possono essere usati in parti successive del programma.

In questo esempio, il segno uguale ha lo stesso significato che in algebra. Ma non è sempre così. Consideriamo ad esempio il seguente segmento di programma:

```
10   LET A = 10.3
20   LET B = 5.6
30   LET C = A+B
40   LET A = C+2*B                                           (2-43)
```

L'azione dei primi tre statements è esattamente la stessa che nel segmento di programma (2-42). Esaminiamo ora lo statement 40; esso fa sì che un valore venga calcolato e memorizzato nella cella di memoria denominata A. Allorché un valore è memorizzato in una cella di memoria, qualsiasi vecchio valore che vi era memorizzato è cancellato. Quindi, in seguito all'esecuzione dello statement 40, il valore memorizzato nella cella di memoria denominata A sarà 10.3+5.6+2(5.6) = 27.1. Se A è usata di nuovo nel resto del programma, il suo valore numerico sarà 27.1.

Per il modo in cui il segno uguale è interpretato nel BASIC, si verificherà che alcuni statements che sono validi in BASIC, e che come tali sono spesso usati, non sono riportabili ad equazioni algebriche. Consideriamo ad esempio il seguente segmento di programma:

```
10  LET B = 2.5
20  LET C = 2*B+3
30  LET B = B+1
```
(2-44)

Analizziamo lo statement 30. L'equazione algebrica corrispondente b = b+1 sarebbe errata. In BASIC, invece, questa espressione significa: prendi il valore memorizzato nella cella di memoria denominata B, somma 1 a questo valore, e memorizza la somma nella stessa cella di memoria denominata B. Quindi, in seguito all'esecuzione dello statement 30, nella cella di memoria B sarà memorizzato il valore 3.5. Si noti che, al momento dell'esecuzione dello statement 20, per B è utilizzato il valore 2.5: solo dopo l'esecuzione dello statement 30 il valore di B diventa 3.5.

Nei segmenti di programma (2-43) e (2-44) abbiamo cambiato il valore memorizzato di una variabile. Ci si può chiedere perché non abbiamo definito una nuova variabile. Ad esempio, nel segmento di programma (2-44) avremmo potuto scrivere lo statement 30 in questo modo:

```
30  LET D = B+1
```
(2-45)

e poi usare D, invece di B, nel resto del programma. Ci sono diversi vantaggi nel ridefinire le variabili, quando i vecchi valori non dovranno più essere usati: in primo luogo, il programma non è costretto a tener presenti tanti nomi di variabili, cosa che, in programmi lunghi, può essere un problema; poi, fatto ancora più importante, la ridefinizione delle variabili risparmia spazio di memoria. Ad esempio, il segmento di programma (2-44) richiede due celle di memoria, una per B, ed un'altra per C: se modifichiamo la linea 30 del segmento (2-44) come nello statement (2-45), saranno necessarie tre celle di memoria. Il numero di variabili che può venir memorizzato è limitato dalle dimensioni della memoria, e, nel caso di grossi programmi, questo fattore è spesso importante. Anzi, su determinati calcolatori non possono essere eseguiti programmi che superano una certa lunghezza.

2-6. PRECISIONE. ESATTEZZA. SUPERAMENTO DELLA CAPACITÀ ARITMETICA

La maggior parte dei calcolatori numerici che utilizzano il BASIC non vanno oltre una precisione di otto o nove cifre significative. (Consultando il manuale BASIC del vostro calcolatore, saprete il numero di cifre significative ammesso.)

Illustriamo l'importanza di questo punto. Supponiamo che siano ammesse otto cifre significative: sembrerebbe che questo fornisca un grado di precisione sufficiente per quasi tutti i calcoli. Ad esempio, consideriamo il numero 1.2678164: un errore eventualmente dipenderebbe dal fatto che le cifre non sono sufficienti per rappresentare la nona cifra significativa, che avrebbe un peso dell'ordine dei dieci miliardesi-

mi. Tuttavia si possono avere delle inesattezze maggiori di questa. Calcoliamo ad esempio a-b, dove a = 100 e b = 99. Il risultato è 1. Se però si ha un errore dell'1% in a, per cui a = 101, allora a-b = 2, che è un errore del 100%: cioè l'esattezza di un calcolo è a volte di gran lunga minore dell'esattezza dei numeri usati nel calcolo. Questi errori sono detti errori *di arrotondamento*. In programmi complessi si ha spesso un numero enorme di calcoli: in tal modo gli errori di arrotondamento si accumulano, aumentando ad ogni nuovo calcolo. Di conseguenza, si possono ottenere risultati errati dovuti agli errori di arrotondamento, anche se il programma non contiene errori. In questi casi non viene stampato alcun messaggio di errore. Pertanto, bisognerebbe stare molto attenti, e verificare sempre i programmi complessi con valori noti, per accertarsi di aver ottenuto dei risultati esatti.

Sembrerebbe che il massimo numero permesso di cifre significative debba porre dei limiti alla grandezza massima dei numeri su cui si può lavorare. In realtà non è così. Vediamo perché. Supponiamo di avere a disposizione otto cifre significative. Usandole tutte, non possiamo scrivere numeri maggiori di

99999999

o minori di

−99999999

Comunque, usando la notazione introdotta nel paragrafo 1-3, possiamo scrivere numeri più grandi. Ad esempio,

9,568,130,000,000 = 9.56813E12

In altre parole, possiamo esprimere numeri molto grandi senza usare un gran numero di cifre significative. Allo stesso modo si possono rappresentare numeri molto piccoli. Ad esempio,

.0000000000136285 = 1.36285E−11

Il valore del numero più grande e di quello più piccolo su cui si può lavorare non è illimitato: tipicamente, i numeri consentiti vanno da 10^{38} a 10^{-38}, compreso lo zero. Naturalmente si possono usare sia i numeri positivi che i numeri negativi: questi vanno da -10^{-38} a -10^{38}. L'intervallo di numeri che si è detto è tipico, ma non vale per tutti i calcolatori: per sapere qual è quello offerto dal vostro calcolatore, consultate il relativo manuale BASIC.

Vediamo ora un po' di terminologia. Se scriviamo un numero come 1.56281E15, allora 1.56281 è detto *parte frazionaria*, mentre 15 è detto *esponente*. L'esponente è un numero intero positivo o negativo, che va generalmente da −38 a +38.

Di solito, quando si usa la notazione con E, i numeri costituenti la parte frazionaria sono formati al massimo da sei o sette cifre significative. Ad esempio, il numero $4.123456789 \times 10^{23}$ si potrebbe rappresentare, con sei cifre significative, come 4.12346×10^{23}: si noti che l'ultima cifra significativa, 5, è stata arrotondata a 6.

Se, durante un calcolo, si ottiene un numero più grande del massimo ammesso dal calcolatore, allora si ha ciò che è definito un *superamento, verso l'alto, dei limiti aritmetici* (*overflow* in inglese). In questi casi viene stampato un messaggio di errore. Il messaggio sarà della forma

% OVERFLOW IN LINE 20

Questo messaggio segnala due cose: che si è verificato un superamento, verso l'alto, dei limiti aritmetici, ed in quale punto (numero di statement) del programma si è verificato. Nel caso che il calcolo prosegua, il numero che ha superato il limite aritmetico superiore sarà sostituito con il massimo numero con cui il calcolatore può lavorare. Si tenga presente che questa procedura varia da calcolatore a calcolatore: in alcuni calcolatori il superamento dei limiti aritmetici provoca la fine dell'elaborazione.

Se, durante un calcolo, si ottiene un numero più piccolo del minimo ammesso dal calcolatore, allora si ha ciò che è definito un *superamento, verso il basso, dei limiti aritmetici* (*underflow* in inglese). Questo è analogo al superamento verso l'alto: in genere il calcolo prosegue con il numero sostituito da 0, a meno che questo comporti la divisione per 0, il che generalmente fa terminare l'elaborazione. Viene generato inoltre un messaggio di errore.

Quando si sommano o si sottraggono numeri molto grandi o molto piccoli, può verificarsi una perdita di esattezza non immediatamente evidente. Supponiamo ad esempio di voler sommare

A = 1.78611E15

e

B = 1.21161E12

Quando si esegue una somma, i numeri da sommare vengono manipolati in modo che gli esponenti siano uguali. Nell'esempio, il secondo numero verrà pareggiato in modo che il suo esponente diventi 15. Avremo così:

1.21161E12 = 0.00121161E15

Ma, dato che sono ammesse solo sei cifre significative, e che in questo caso i tre

zeri sono considerati cifre significative, allora le ultime tre cifre vengono arrotondate. Di conseguenza, il valore di B che è sommato ad A è

0.00121E15

Si è avuta dunque una perdita di esattezza. Ancora una volta, occorre verificare, servendosi di valori tipici, se nel programma compaiono errori di arrotondamento apprezzabili.

ESERCIZI

2-1. Determinate i valori che si ottengono per X, Y ed A come risultato del seguente segmento di programma scritto in BASIC:

```
10   LET A = 15.3
20   LET B = 2.63
30   LET C = –29.4
40   LET D = 1.532
50   LET X = A–B + C–D
60   LET Y = A + B + C + X
70   LET Z = 15 + Y + X + A
```

Verificate i risultati completando il programma ed eseguendolo su di un calcolatore.

2-2. Scrivete uno statement BASIC equivalente alla seguente espressione algebrica:

$$x = a+b+c+d$$

Verificate il risultato scrivendo un semplice programma in BASIC ed eseguendolo su di un calcolatore. I valori da usare sono: $a = 1.5$; $b = 0.3$; $c = 2.7$; $d = 3.8$.

2-3. Ripetete l'Esercizio 2-2 con

$$x = a+b-c-d$$

2-4. Ripetete l'Esercizio 2-1, ma supponendo che gli statements 50, 60 e 70 siano sostituiti da

```
50   LET X = A*B*C*D
60   LET Y = A/B
70   LET Z = A*B*C/D
```

2-5. Scrivete un programma in BASIC che calcoli le seguenti equazioni algebriche:

$$x = ab + cd + a/b$$
$$y = a/b + ac/d$$
$$z = -xy/a$$

I valori da usare sono: $a = 15.3$; $b = -16.7$; $c = 23.4$; $d = 2.3$. Eseguite il programma su di un calcolatore.

2-6. Ripetete l'Esercizio 2-5 con le equazioni algebriche

$$x = a^2 + c^b + d/c$$
$$y = a^2 + b^3 + c^d ab$$
$$z = abc^d + c^d + bc^2$$

usando i valori dati nell'Esercizio 2-5. Provate ad eseguire questo programma su di un calcolatore.

2-7. Quale problema vi si presenterebbe, se cercaste di scrivere un programma in BASIC per calcolare

$$x = ab + b^d + ca$$

usando i valori dati nell'Esercizio 2-5? Provate ad eseguire questo programma su di un calcolatore.

2-8. Determinate i valori che si otterranno per X, Y, Z, X1, Y1 e Z1 come risultato del seguente segmento di programma in BASIC:

```
10  LET A = 3
20  LET B = 2
30  LET C = -2
40  LET D = 3.41
50  LET X = A*B+C/D*A
60  LET Y = A + B*C/D-A*B
70  LET Z = A/B*C + D*B↑3
80  LET X1 = A/B*C + D*A↑B + A*B
90  LET Y1 = A*B*C + B↑A↑C
100 LET Z1 = A*B↑2*3 + C↑4-D↑A
```

Verificate i risultati completando il programma ed eseguendolo su di un calcolatore.

2-9. Ripetete l'Esercizio 2-8 con il seguente segmento di programma:

```
 10  LET A = 2.36
 20  LET B = 1.23E2
 30  LET C = –3.21E–3
 40  LET D = 3
 50  LET E = 2
 60  LET F = –2
 70  LET X = –A↑D + B↑E↑F*D
 80  LET Y = A + D*2↑3 + A + B*C↑F
 90  LET Z = –A↑2↑E–D*E*F↑2 + A + B/C
100  LET X1 = B*A/D↑E + A/B↑F + B*C*D/F
110  LET Y1 = B + X1/E
120  LET Z1 = X1 + Y1*2*E↑D
```

2-10. Ripetete l'Esercizio 2-9 con gli statements da 70 a 120 sostituiti dagli statements seguenti:

```
 70  LET X = (A + B)*(C + D)
 80  LET Y = (A + 2 + C)*(D*E–F)
 90  LET Z = (A + B*C)*(D*B + C*D)/(A + F*B)
100  LET X1 = (A + B)↑(D–E) + (C*F–A)/(B + D↑E)
110  LET Y1 = (A + B + C*F↑2)/(C↑2 + A*B)*(A + B)
120  LET Z1 = (X1 + Y1)*(D + E) + (X1–Y1)↑E
```

2-11. Ripetete l'Esercizio 2-9 con gli statements da 70 a 120 sostituiti dagli statements seguenti:

```
 70  LET X = (A + (B + C)*(D + E) + E/F)*(A + B*(D + F))
 80  LET Y = (A + D)/(B + (C + E)/(C + F))↑2
 90  LET Z = (A–B↑(E + 1) + B*A)↑(D + E + F)↑2
100  LET X1 = (A + B + C)*(A–B*(C + D) + (E–F)/C)/(A + B)↑D
110  LET Y1 = (X1 + Z)*(X1*(X1 + X)/((X + Y)*X + Z))
120  LET Z1 = (X1 + Y1)↑(3–D)
```

2-12. Ripetete l'Esercizio 2-9 con gli statements da 70 a 120 sostituiti dagli statements seguenti:

```
 70  LET X = (A + (B + (C + D)*(E + F))
     *D/(A–(C + D)*(E + F)))↑E
 80  LET Y = (A + (B + C)↑(2 + E*(D + F)))
     *((A + (B + C)*(D + F)) + B(C + D))
 90  LET Z = (((A + B(C + D) + F*(A–B))*(A–B))*(D–F))*E↑2
100  LET X1 = (A + ((B + C)/(C + D))*(A–B))/(A + B*C↑2)
110  LET Y1 = ((X1 + Z)*(X/(Y + Z + X1(A + B))))*(A–B)
120  LET Z1 = ((X1 + Y1*(A + B)/(C + D))/(A–B)/(X + Y))↑(E–F)
```

2-13. Scrivete un programma in BASIC che calcoli le seguenti espressioni algebriche:

$$x = a + b^2 + c^d$$
$$y = a - ba^2 + c + da^b$$
$$z = b + c/a^2 - c + d - a/b^2$$

I valori da usare sono: a = 1.3; b = 3; c = 3.7; d = 2. Verificate il programma eseguendolo su di un calcolatore.

2-14. Scrivete un programma in BASIC che calcoli le seguenti espressioni algebriche:

$$x = (a+b)(c+d)+(a-d)/(d-b)$$
$$y = (a+c^2)/(b+c^d) - ac(b+c^2)$$
$$z = a - cb^{d+1} - b(a+c)/(a-b^d)$$

Usate i valori di a, b, c e d dati nell'Esercizio 2-13. Verificate il programma eseguendolo su di un calcolatore.

2-15. Ripetete l'Esercizio 2-14 con

$$x = (a + b)(c + b^{(b+1)})/[a-b(a + bc^d)]$$
$$y = (a-b)^2(c^2 + d^2)/[(a-bc^d)(a + b^2c^2)]$$
$$z = [(a^2-b^2c^2)(c + d^d) + a] / [(b + c^2)-9a + b^dc)(a)]$$

2-16. Ripetete l'Esercizio 2-14 con

$$x = \{(a + b^d) + [c + b^{-(d+1)} + c(a + b)]\} / [(a + b)(c + d)^2]$$
$$y = \{[a + (bc)^2 + 2] + [b-c(a + b^d)^{(d-b)}]\} [(a + b + x)(x + 1)]$$
$$z = \{(x + y) - [a + (b + c)(x-y]\}^{b+d+1}$$

2-17. Perché in uno statement aritmetico BASIC deve comparire un egual numero di parentesi aperte e chiuse?

2-18. Dopo che i seguenti statements BASIC saranno stati eseguiti, quali saranno i valori di A, B e C presenti in memoria?

```
10  LET A = 1
20  LET B = 2
30  LET C = A + B
40  LET A = C*B
50  LET B = A + C
```

2-19. Ripetete l'Esercizio 2-18 con il seguente segmento di programma:

```
10  LET A = 1
20  LET B = 2
30  LET C = (A + B)*2*B
40  LET B = C + A
50  LET A = B + A
```

2-20. Ripetete l'Esercizio 2-18 con il seguente segmento di programma:

```
10  LET A = 1
20  LET B = 2
30  LET C = (A + B)*2*B
40  LET B = A + B*C
50  LET B = 1
60  LET A = A + 1
```

2-21. Dite quante cifre significative sono previste sul vostro calcolatore per i numeri interi e per i numeri rappresentati con la notazione con E.

2-22. Parlate dell'errore di arrotondamento. Perché l'esattezza di un calcolo si può considerare minore di quella dei singoli numeri adoperati in quello stesso calcolo?

2-23. Quale risultato otterrete sommando i numeri seguenti?

1.23456E10
2.34567E3

Supponete di disporre di sette cifre significative.

2-24. Scrivete un programma in BASIC che converta una lunghezza espressa in pollici in una espressa in centimetri.

1 pollice = 2.54 cm.

Verificate il programma eseguendolo su di un calcolatore.

2-25. Scrivete un programma in BASIC che converta in centimetri una lunghezza espressa in piedi, pollici ed 1/16 di pollice. Verificate il programma eseguendolo su di un calcolatore.

2-26. La somma di n numeri 1 + 2 + 3 + ... + n è data da n(n + 1)/2. Scrivete un programma in BASIC che calcoli questa somma. Il valore di n va fornito come dato. Verificate il programma eseguendolo su di un calcolatore più volte, con valori di n ogni volta diversi.

2-27. Scrivete un programma in BASIC che calcoli la circonferenza e l'area di un cerchio il cui raggio viene fornito come dato. Verificate il programma eseguendolo su di un calcolatore più volte, con valori del raggio ogni volta diversi.

2-28. Una persona acquista una casa ed ottiene un'ipoteca della durata di y anni. L'ammontare del mutuo è p. Il tasso d'interesse sull'ipoteca è i. La rata da pagare mensilmente è data da

$$m = \frac{p\frac{i}{12}(1 + \frac{i}{12})^{12y}}{(1 + \frac{i}{12})^{12y} - 1}$$

Scrivete un programma che calcoli le rate mensili, la somma totale pagata allo scadere di sette anni, e l'interesse totale pagato allo scadere di y anni (differenza fra p e la somma totale pagata). I dati sono p, i ed y. Verificate il programma eseguendolo su di un calcolatore. Confrontate i differenti risultati ottenuti per un mutuo p = L. 20.000.000, con un tasso d'interesse del 15.5% e del 18.5%, e con un'ipoteca calcolata su 25 e su 35 anni.

CAPITOLO 3

STATEMENTS DI INGRESSO E DI USCITA

Fin qui abbiamo parlato dell'ingresso e dell'uscita dei dati in modo non approfondito. In questo capitolo tratteremo le procedure d'ingresso e di uscita molto più dettagliatamente, e ne evidenzieremo la notevole versatilità. Per maggiore completezza, ritorneremo su alcuni argomenti esposti nei paragrafi 1-2 e 1-3.

3-1. GLI STATEMENTS DI READ E DI DATA

Nel paragrafo 1-3 abbiamo illustrato l'uso degli statements di READ e di DATA. Ora riassumiamo, ampliandolo, l'argomento. Il semplice programma che segue illustra l'uso di questi statements:

```
10  REM SEMPLICE PROGRAMMA ILLUSTRANTE
15  REM GLI STATEMENTS DI READ E DI DATA
20  READ A,B,C
30  LET X = A + B + C
40  PRINT X,A,B,C
50  DATA 10,20,30
60  END                                        (3-1)
```

Gli statements di READ e di DATA si usano insieme: i valori numerici per le variabili dello statement di READ sono forniti dallo statement di DATA. Pertanto, nel programma (3-1), alle variabili verranno assegnati, in base agli statements 20 e 50, i seguenti valori:

A = 10
B = 20
C = 30

La forma dello statement di READ è: numero di statement, la parola-chiave READ seguita da una serie di variabili separate da virgole. Si osservi che non c'è la virgola né dopo la parola READ, né alla fine dello statement. La forma dello statement di DATA è: numero di statement , la parola-chiave DATA seguita da una serie di numeri separati da virgole. Si noti che anche qui non c'è la virgola né dopo la parola DATA né alla fine della linea. (Nel Capitolo 8 parleremo dell'ingresso di quantità diverse dai numeri.)

Nel programma (3-1) le variabili dello statement di READ ed i numeri dello statement di DATA si corrispondono esattamente. Quindi il primo numero dello statement di DATA è assegnato alla prima variabile dello statement di READ, e cosi via. Comunque questa esatta corrispondenza non è indispensabile. Illustriamo questo concetto con un segmento di programma che contiene diversi statements di READ e di DATA:

```
        .
        .
        .
 20   READ A,X
 30   READ Y
 40   READ Z,B
        .
        .
        .
100   DATA 22.1, 15.6, 24.3
110   DATA 18.9
120   DATA 17.5, 19.3
999   END                                    (3-2)
```

I puntini in sequenza verticale indicano altri statements del programma; si suppone inoltre che nella parte non riportata del programma non ci siano altri statements di READ e di DATA. Tutti gli statements di DATA, nel loro insieme, si dice che formano un *blocco di dati*: questo può essere considerato come un elenco di tutti i dati del programma. I numeri sono elencati nell'ordine in cui compaiono allorquando gli statements di DATA sono ordinati secondo il numero di statement. Per il programma (3-2), abbiamo:

```
22.1
15.6
24.3
18.9
17.5
19.3                                         (3-3)
```

All'inizio dell'esecuzione, vi è un *puntatore* (immaginario) che indica il primo numero del blocco di dati: quando viene eseguito uno statement di READ, il numero così indicato viene letto. Letto un numero, il puntatore avanza fino al numero successivo. Allora, quando è eseguito lo statement 20 del programma (3-2), ad A viene assegnato il valore 22.1, che è il primo numero della lista; poi il puntatore avanza fino al secondo numero della lista di dati, cosicché il valore 15.6 viene assegnato ad X. Ora il puntatore si muove fino al terzo numero della lista, e vi si ferma fino a quando è eseguito lo statement 30: a questo punto 24.3 è assegnato ad Y, e quindi il puntatore avanza fino al quarto numero della lista. La procedura procede finché sono eseguiti tutti gli statements di READ. Concludendo, all'esecuzione del programma (3-2) alle variabili saranno assegnati i seguenti valori:

A = 22.1
X = 15.6
Y = 24.3
Z = 18.9
B = 17.5 (3-4)

Si osservi che nel programma (3-2) sono stati specificati troppi dati: ci sono infatti cinque variabili e sei numeri. I dati in più vengono semplicemente ignorati.

Una situazione totalmente diversa si verifica quando i dati forniti non sono in numero sufficiente. In questo caso viene stampato un messaggio di errore, e l'elaborazione termina. Ad esempio, supponiamo che nel programma (3-2) le linee 110 e 120 siano state omesse; in fase di esecuzione del programma, verrà stampato il seguente messaggio di errore:

```
% OUT OF DATA IN 40
```

e l'elaborazione s'interromperà. Il messaggio di errore segnala che non c'è un numero sufficiente di dati da assegnare alle variabili contenute nello statement di READ numero 40. Si noti che l'elaborazione s'interrompe allo statement 40, ma che, nondimeno, tutti gli statements precedenti verranno eseguiti. Se, prima del punto in cui il programma si trova senza dati, ci sono degli statements di PRINT, questi verranno eseguiti: quindi questa parte di programma fornirà dei risultati, prima che venga stampato il messaggio di errore e che l'elaborazione si arresti.

Non è necessario che gli statements di READ e di DATA siano presenti in egual numero. Ad esempio, il seguente segmento di programma è equivalente al segmento (3-2):

```
.
20  READ A,X
```

```
30  READ Y
40  READ Z,B
     .
     .
     .
110  DATA 22.1, 15.6, 24.3, 18.9, 17.5, 19.3
999  END                                          (3-5)
```

Le stesse regole sono usate per leggere i dati, per cui le assegnazioni dei dati alle variabili sono le medesime: un altro segmento di programma equivalente al (3-2) ed al (3-5) per quanto riguarda l'assegnazione dei dati è

```
     .
     .
     .
20  READ A,X,Y,Z
30  READ B
     .
     .
     .
110  DATA 22.1, 15.6, 24.3, 18.9, 17.5, 19.3
999  END                                          (3-6)
```

3-1-1. Collocazione dello statement di DATA

Nei precedenti esempi lo statement di DATA compariva alla fine del programma, subito prima dello statement di END. Ma non è necessario che sia sempre così: gli statements di DATA possono essere situati in qualunque punto del programma. Comunque è *buona regola di programmazione* collocare gli statements di DATA alla fine: così sono facilmente rintracciabili e modificabili, quando si desidera cambiare i dati. Attenzione: se ci si dimentica, in un grosso programma, di uno statement di DATA, può capitare di assegnare valori errati alle variabili. Un'altra buona ragione per collocare gli statements di DATA alla fine del programma è che spesso, nello scrivere un programma, ci si accorge di aver bisogno di ulteriori dati. In questo caso si può ricorrere ad ulteriori statements di READ e aggiungere uno statement di DATA senza dover rinumerare le linee.

3-2. LO STATEMENT DI RESTORE

A volte occorre leggere gli stessi dati più di una volta. Consideriamo ad esempio il seguente segmento di programma:

```
10   READ X,Y
     .
50   READ A,B,C,D
     .
100  DATA 2, 3
110  DATA 2, 3, 15.6, 19.8
```
(3-7)

All'esecuzione dello statement 10 i valori 2 e 3 sono assegnati ad X ed Y rispettivamente; all'esecuzione dello statement 50 i valori 2, 3, 15.6 e 19.8 sono assegnati ad A, B, C e D rispettivamente. Supponiamo che i valori da assegnare ad X e ad A siano sempre gli stessi, e che siano gli stessi anche i valori da assegnare ad Y e a B: allora dovremo introdurre due volte gli stessi dati. Possiamo eliminare la necessità di introdurre gli stessi dati due volte ricorrendo allo statement di RESTORE. Vediamo come si procede. Consideriamo il segmento di programma

```
10   READ X,Y
     .
49   RESTORE
50   READ A,B,C,D
     .
100  DATA 2,3,15.6,19.8
```
(3-8)

Questo porterà alla stessa assegnazione di dati del segmento di programma (3-7): all'esecuzione dello statement di RESTORE, il puntatore viene arretrato alla prima linea del blocco di dati, cosicché ad A e a B vengono assegnati rispettivamente i valori 2 e 3. Cioè, in seguito all'esecuzione dello statement 10, il puntatore indica il terzo elemento, 15.6, ma all'esecuzione dello statement di RESTORE il puntatore è ricollocato al primo elemento del blocco di dati (cioè 2). Pertanto i valori assegnati ad A, B, C e D sono 2, 3, 15.6, e 19.8 rispettivamente. (Si tenga presente che il puntatore ritorna al primo elemento del blocco di dati indipendentemente dal numero degli statements di DATA.)

Voi forse chiederete perché ci si prenda la briga di far questo! Invece di leggere A, B, C e D nello statement 50, perché non limitarsi a leggere C e D e ad usare X ed Y al posto di A e B nel resto del programma? Si osservi che (v. par. 2-5) può essere

necessario cambiare il valore di una variabile per risparmiare memoria! Consideriamo ad esempio il programma seguente:

```
 10   READ A,B
 20   LET A = (A*2)*B
 30   LET B = B↑2 + 3
 40   PRINT A,B
 50   RESTORE
 60   READ A,B,C,D
 70   LET A = A↑2 + B*C + D
 80   PRINT A
 90   DATA 1.6, 2.3, 4.7, 5.8
100   END                                          (3-9)
```

I valori di A e B sono letti ed impiegati nei calcoli degli statements 10-40. Si osservi che vengono utilizzate due sole celle di memoria: naturalmente i valori di A e B cambiano durante l'esecuzione di questo statement. A questo punto è eseguito lo statement di RESTORE, e il puntatore ritorna al primo elemento del blocco di dati. Ora, all'esecuzione dello statement 60, i valori originari di A e B possono essere letti nuovamente ed utilizzati in ulteriori calcoli. Il programma (3-9) sfrutta soltanto quattro celle di memoria per le variabili, perché le celle di memoria destinate ad A e B vengono utilizzate.

Potremmo ottenere lo stesso effetto anche non ricorrendo allo statement di RESTORE, solo che in questo caso dovrebbero essere inserite due nuove voci di dati. Ma se un programma va eseguito spesso con dati diversi, questo si risolverebbe in lavoro extra, soprattutto quando interviene un gran numero di dati. Quindi, concludendo, l'uso dello statement di RESTORE semplifica l'esecuzione del programma.

Ci sono dei casi in cui non si desidera riposizionare il puntatore all'inizio del blocco di dati: supponiamo ad esempio di volerlo riportare al secondo elemento del blocco di dati. Lo statement di RESTORE riporta sempre e comunque il puntatore al primo elemento, ma, con opportuni accorgimenti di programmazione, possiamo "aggirare" il problema. Consideriamo il seguente segmento di programma:

```
 10   READ X, Y
        .
 50   RESTORE
 60   READ A
 70   READ A,B,C
        .
100   DATA 1,2,3,4                                 (3-10)
```

Vogliamo assegnare i valori 1 e 2 ad X ed Y rispettivamente, e i valori 2, 3 e 4 ad A, B e C rispettivamente. All'esecuzione dello statement 10, i valori 1 e 2 saranno assegnati ad A e B. Lo statement 50 riposiziona il puntatore al primo elemento del blocco di dati. All'esecuzione dello statement 60, il valore 1 è assegnato ad A, e *il puntatore avanza fino al secondo* elemento del blocco di dati. Ora, all'esecuzione dello statement 70, ad A sarà assegnato il valore 2, che sostituirà il valore 1 memorizzato nella cella di memoria A. L'esecuzione dello statement 70 fa anche sì che i valori 3 e 4 siano assegnati a B e C. Si noti che lo statement 60 è uno statement di READ *fittizio*, il cui unico scopo è di far avanzare il puntatore. La variabile che viene letta nello statement 60 non è mai usata, e potremmo denominarla come vogliamo, purché non venga mai usata in seguito. Comunque, chiamandola A e leggendo di nuovo A, risparmiamo spazio di memoria. Si tenga presente che, volendo far avanzare il puntatore di più di uno "spazio" dopo l'esecuzione dello statement di RESTORE, occorre inserire delle altre variabili nello statement di READ fittizio.

3-3. LO STATEMENT DI INPUT

Quando si usano gli statements di READ e di DATA, i dati vengono introdotti in fase di scrittura del programma; se è necessario utilizzare nuovi dati, bisogna modificare il programma. Esiste un diverso modo di operare, vantaggioso soprattutto con calcolatori che lavorano in timesharing: in questo caso, è durante l'esecuzione stessa del programma che il calcolatore "chiede" all'operatore di introdurre dati. In altre parole, durante l'esecuzione del programma il calcolatore sollecita l'utilizzatore, che in risposta batte da terminale tutti i dati necessari; quindi l'esecuzione prosegue fino all'uscita del risultato. In questo modo si possono facilmente cambiare i dati ogni volta che il programma viene eseguito.

Per introdurre dati secondo questa modalità usiamo uno statement individuato dalla parola-chiave INPUT. Illustriamone l'uso con un breve programma:

```
10  REM PROGRAMMA ILLUSTRANTE
15  REM LO STATEMENT DI INPUT
20  INPUT A,B,C
30  LET = A+B+C
40  PRINT D
50  END                                    (3-11)
```

Ora, immaginiamo che il programma sia stato introdotto nel modo illustrato nel paragrafo 1-4: allora, quando il calcolatore invia il carattere di sollecitazione (> tipicamente), battiamo RUN seguita dal ritorno carrello.

```
> RUN
```

Supponiamo che il nome del programma sia ABC. Il calcolatore risponde con

```
ABC         12:30         19-MAR-77
?
```

Il punto interrogativo segnala che sono richiesti dei dati: supponiamo che l'utente sappia che sono necessari tre valori (nel paragrafo 3-5 vedremo come si possano stampare informazioni ulteriori che dicano all'utente quanti dati esattamente sono richiesti). A questo punto l'utilizzatore risponde con tre numeri, seguiti dal ritorno carrello:

```
? 5,10,20
```

(Si osservi che ogni numero è separato dal successivo mediante una virgola.) Allora il calcolatore invia la risposta:

```
35
```

Ricapitolando, la sequenza completa per eseguire il programma sarà:

```
> RUN
ABC         12:30         19-MAR-77
? 5,10,20
35
>
```

Le uniche parole battute dall'utilizzatore sono RUN e i dati 5, 10, 20. Ognuna di queste linee è conclusa da un ritorno carrello.

Lo statement di INPUT elenca un certo numero di variabili. Ad esempio, nel programma (3-11), sono elencate tre variabili: questo implica che almeno tre numeri, separati da virgole, debbono essere introdotti da terminale in risposta al punto interrogativo di richiesta di dati. I dati in più sono ignorati; se invece viene battuto un numero di dati inferiore a quello richiesto, il calcolatore risponde con uno statement che richiede ulteriori dati, e l'elaborazione non procede finché non si siano forniti tutti i dati dichiarati nello statement di INPUT.

Quando si scrive lo statement di INPUT, ciascuna variabile (tranne l'ultima) dev'essere separata dalla successiva mediante una virgola.

In un programma può comparire più di uno statement di INPUT, ciascuno dei quali genererà un ? durante l'esecuzione del programma. Per ciascuno statement bisogna introdurre i dati appropriati: ovvero, le osservazioni che abbiamo svolto per

un programma contenente un solo statement di INPUT valgono per ciascuno degli statements di INPUT presenti nel programma. Illustriamo con un esempio questa situazione:

```
10   REM PROGRAMMA CON VARI
15   REM STATEMENTS DI INPUT
20   INPUT A,B
30   INPUT C
40   LET D = A + B + C
50   PRINT D
60   END                                                    (3-12)
```

L'esecuzione di questo programma sarà la seguente:

```
> RUN
ABC          10:15          15-FEB-77
? 10,1.5E2
? 2.0
162
>
```

La parola RUN e le due linee di dati vengono battute dall'utilizzatore; il resto è scritto dal calcolatore. Si noti che il primo ? è stampato in seguito all'esecuzione dello statement 20, e sollecita l'introduzione di due numeri, come dati; il secondo punto interrogativo è stampato in seguito all'esecuzione dello statement 30, e in questo caso è richiesta l'introduzione di un solo numero.

Si tenga presente che i numeri introdotti possono essere numeri interi, decimali, o numeri espressi nella notazione con E: in altre parole, si possono usare tutti i metodi di cui il BASIC si avvale per rappresentare i numeri.

3-4. LO STATEMENT DI PRINT

Quando vogliamo stampare dei dati, usiamo lo statement di PRINT (v. par. 1-2). Ad esempio,

```
90   PRINT A,B,X1,X2                                        (3-13)
```

Lo statement di PRINT ha la forma: numero di linea, la parola-chiave PRINT seguita da un elenco di variabili. Le variabili sono separate da virgole; non c'è virgola dopo PRINT e alla fine dell'elenco di variabili. Più avanti, in questo stesso paragrafo,

illustreremo le modifiche che si possono apportare a parecchi di questi stetements; per il momento, consideriamo questa come la forma dello statement di PRINT.

Analizziamo l'effetto dello statement di PRINT. Supponiamo che, prima dell'esecuzione dello statement 90, siano stati assegnati alle variabili i seguenti valori:

$$\begin{aligned} A &= 23 \\ B &= 1567.3014 \\ X1 &= 1.26712E-10 \\ X2 &= -176 \end{aligned} \tag{3-14}$$

Come risultato dell'esecuzione dello statement (3-13), sarà stampata la seguente uscita:

```
23          1567.3014       1.26712E-10      -176
```

Si osservi come i numeri sono spaziati sulla pagina. Soffermiamoci su questo punto. Ciascuna linea è divisa in *zone*, o regioni, e ciascuna zona è formata da 15 spazi. Normalmente su una linea di telescrivente ci sono cinque zone. I dati vengono stampati in una data zona a partire dal margine sinistro della zona stessa. Una virgola situata nello statement di PRINT fa sì che la testina di stampa della telescrivente avanzi fino alla zona successiva. Ciò premesso, esaminiamo lo statement di PRINT (3-13). Per primo è stampato, a partire dalla prima colonna, il valore numerico di A; poi si incontra la virgola, per cui la testina di stampa avanza fino allo spazio 16, a partire dal quale viene stampato il valore numerico di B. A questo punto nello statement di PRINT s'incontra un'altra virgola: la testina di stampa è avanzata fino allo spazio 31, dove viene stampato il valore numerico assegnato ad X1. S'incontra ancora un'altra virgola, che posiziona la testina di stampa sullo spazio 46: è allora stampato il valore numerico assegnato ad X2. Si tenga presente che, se il numero è negativo, il segno meno è stampato sul margine sinistro della zona. Una volta stampate tutte le variabili specificate nello statement di PRINT, la testina di stampa è riportata alla colonna 1, e la carta avanza di una linea.

Se lo statement di PRINT contiene più di cinque variabili, queste non entrano tutte in una sola linea: in tal caso i dati eccedenti sono stampati sulla linea successiva. Se poi nello statement di PRINT sono presenti più di dieci variabili, i dati saranno stampati su tre linee, e così via. Nel seguente esempio:

```
110  PRINT A,B,C,D,E,F,G,H                                   (3-15)
```

dove, prima dell'esecuzione di questo statement, alle variabili sono stati attribuiti i valori A = 1, B = 2, C = 3, D = 4, E = 5, F = 6, G = 7, H = 8, all'esecuzione dello statement di PRINT si avrà una stampa siffatta:

```
1       2       3       4       5
6       7       8
```

Si può allargare la spaziatura fra i numeri stampati inserendo delle altre virgole nello statement di PRINT. Supponiamo ad esempio di avere:

```
130   PRINT A, , B,C                                    (3-16)
```

dove i valori assegnati alle variabili sono A = 1, B = 2, C = 3. Allora, la linea stampata sarà:

```
1                 2         3
```

Quando lo statement viene eseguito, A è stampata a partire dallo spazio 1; poi s'incontrano due virgole, che fanno avanzare la testina di stampa di due zone fino allo spazio 31, nel quale viene stampata B. S'incontra poi una sola virgola, per cui ci si posiziona sullo spazio 46, dove viene stampata C.

3-4-1. Il formato compatto

In alcuni casi la spaziatura fra i dati che sfrutta zone di 15 caratteri è troppo larga. Si può ottenere una spaziatura minore sostituendo nello statement di PRINT alle virgole dei punti e virgola. L'effetto del punto e virgola varia da calcolatore a calcolatore. Tipicamente, quando s'incontra un punto e virgola in luogo di una virgola, la larghezza della zona si riduce: la larghezza delle zone ridotte dipenderà dal numero massimo di cifre nei valori assegnati alle variabili. La Tabella 3-1 riporta la spaziatura più usuale delle zone.

Cifre del numero	Spazi della zona
1 - 3	6
4 - 6	9
7 - 9	12

Tabella 3.1 — Larghezza delle zone quando si usa il punto e virgola.

Consideriamo il seguente statement:

```
100   PRINT A;B;C;D;E;F;G;H                             (3-17)
```

dove sono stati assegnati i seguenti valori: A = 1, B = 2, C = 3, D = 4, E = 5, F = 6, G = 7, H = 8. All'esecuzione dello statement di PRINT verrà stampato il testo che segue:

```
1    2    3    4    5    6    7    8
```

Può darsi che con il vostro calcolatore l'effetto del punto e virgola sia diverso da quello che si è detto qui, ma comunque, in linea generale, aspettatevi una stampa più ravvicinata e ristretta. Al solito, consultate il manuale BASIC del calcolatore utilizzato per conoscere l'esatto effetto del punto e virgola dello statement di PRINT.

3-4-2. Più statements di PRINT in un programma

Se in un programma compare più di uno statement di PRINT, ciascuno di essi sarà eseguito indipendentemente dagli altri, cioè, dopo l'esecuzione di uno statement di PRINT, la testina di stampa ritorna allo spazio 1, e la carta avanza di una linea; se vi è un altro statement di PRINT, la stampante comincia a stampare a partire dallo spazio 1 senza avanzamento della carta. (Vedremo più avanti delle modifiche a questo punto.) Ad esempio, immaginiamo di avere gli statements

```
110  PRINT A,B,C
120  PRINT D,E,F
```
(3-18)

dove sono stati assegnati i seguenti valori: A = 1, B = 2, C = 3, D = D = 4, E = 5, F = 6. Quando gli statements 110 e 120 sono eseguiti, sarà stampato il testo seguente:

```
1     2     3
4     5     6
```

Volendo far comparire una riga bianca fra due righe di stampa, occorre che nello statement di PRINT non si abbiano variabili. Ad esempio,

```
110  PRINT A,B,C
115  PRINT
120  PRINT D,E,F
```
(3-19)

Con i valori precedentemente assegnati, verrà stampato questo testo:

```
1     2     3

4     5     6
```

In generale, uno statement di PRINT senza variabile fa avanzare la carta della telescrivente di una riga tutte le volte che viene eseguito: uno statement di questo tipo può comparire in qualunque punto del programma.

A volte è opportuno inserire in un programma due (o più) statements di PRINT. Supponiamo di voler stampare dei dati come se ci fosse un solo statement di PRINT: se l'ultima variabile contenuta nello statement di PRINT è seguita da una virgola, la testina di stampa avanzerà fino alla zona successiva senza ritornare alla colonna 1. All'esecuzione del successivo statement di PRINT, la stampa ripartirà dal punto in cui si era fermata. Ad esempio, consideriamo il seguente segmento di programma:

```
150   PRINT A,B,
  .
  .
  .
190   PRINT C,D
```
(3-20)

dove sono stati assegnati i seguenti valori: A = 1, B = 2, C = 3, D = 4. (I tre puntini indicano eventuali statements intermedi, nessuno dei quali è uno statement di PRINT.) Allora, l'esecuzione degli statements 150 e 190 darà:

```
1             2             3             4
```

Nella nostra trattazione di un programma con molti statements di PRINT abbiamo usato delle virgole dopo le variabili. Orbene, le virgole possono essere sostituite da punti e virgola, nel qual caso i concetti non cambieranno, ma avremo la forma compatta. Se, ad esempio, agli statements (3-20) sostituiamo

```
150 PRINT A;B;
  .
  .
  .
190 PRINT C;D
```
(3-21)

avremo:

```
1          2          3          4
```

3-4-3. Uso di espressioni aritmetiche negli statements di PRINT

Abbiamo spiegato l'uso dello statement di PRINT per la stampa dei valori numerici delle variabili. In realtà, lo statement di PRINT ha una maggiore versatilità: in esso

possono infatti comparire delle espressioni aritmetiche. In questo caso, all'esecuzione dello statement di PRINT, sarà stampato il valore numerico risultante dal calcolo dell'espressione aritmetica. Vediamo l'esempio seguente:

```
150   PRINT A,B*6*A,C                                    (3-22)
```

dove i valori di A, B e C sono stati in precedenza così fissati: A = 2, B = 3, C = 4. All'esecuzione di questo statement, verrà stampato quanto segue:

```
2            36            4
```

Infatti il valore di B*6*A è 3(6) 2 = 36, e questo numero viene stampato nella seconda zona esattamente come se fosse una singola variabile. In generale, se nello statement di PRINT è contenuta un'espressione aritmetica, questa sarà calcolata, e il risultato verrà stampato esattamente come se nello statement di PRINT comparisse una singola variabile. Tutto quanto si è detto in questo paragrafo è applicabile alle espressioni aritmetiche contenute negli statements di PRINT: così, se invece delle virgole compaiono dei punti e virgola, si avrà una spaziatura più stretta, e così via.

3-5. STAMPA DI UN TESTO

È spesso utile poter stampare un testo associato a dati in uscita. Ad esempio supponiamo di voler ricavare la media dei voti di uno studente e che questa, indicata con la variabile A1, sia uguale a 92.3. Invece di stampare semplicemente

```
92.3
```

sarebbe meglio stampare

```
MEDIA=          92.3                                     (3-23)
```

Questo si può ottenere modificando lo statemente di PRINT. Nel caso dell'esempio precedente, avremo:

```
90   PRINT "MEDIA =",A1                                  (3-24)
```

Questo statement di PRINT farà stampare la linea (3-23). Esaminiamo questo punto: se vogliamo che un certo testo venga stampato, non dobbiamo far altro che

inserirlo fra virgolette nello statement di PRINT. Si osservi che una virgola separa il testo fra virgolette dalla variabile. Virgole e punti e virgola hanno lo stesso significato che nello statement di PRINT senza testo (di cui si è parlato nell'ultimo paragrafo), e cioè, nel caso dello statement (3-24), accade quanto segue: il testo fra virgolette viene stampato e, quando s'incontra la virgola, la testina di stampa si posiziona all'inizio della zona successiva. Nel nostro esempio, poiché MEDIA = occupa meno di 15 spazi, la testina di stampa avanza fino allo spazio 16, a partire dal quale verrà stampato il valore numerico di A. Se, invece della virgola, ci fosse un punto e virgola, si seguirebbe la stessa procedura, ma in questo caso risulterebbe una stampa più compatta (zone più piccole). Ad esempio, sostituendo a (3-24)

```
90  PRINT "MEDIA =";A1                                    (3-25)
```

si avrebbe il testo seguente:

```
MEDIA = 92.3                                              (3-26)
```

In uno statement di PRINT possono trovarsi diversi gruppi di testi, come nel seguente esempio:

```
110  PRINT "MEDIA =",A1,"VOTO" =",G                       (3-27)
```

dove A1 rappresenta la media, e G il voto dell'esame finale. Supponiamo che i valori di A1 e G siano 92.3 e 95 rispettivamente. All'esecuzione di questo statement, verrà stampato quanto segue:

```
MEDIA =        92.3              VOTO =        95          (3-28)
```

Anche qui, si può ottenere una spaziatura più stretta sostituendo le virgole con punti e virgola.

Il testo può venir stampato senza variabili. Vediamo come.

```
110  PRINT "LE RISPOSTE SONO"
120  PRINT "MEDIA =", A1, "VOTO =",G                      (3-29)
```

porta alla stampa di

```
LE RISPOSTE SONO
MEDIA =        92.3              VOTO =        95          (3-30)
```

Vale a dire, se uno statement di PRINT contiene unicamente un testo fra virgolette, soltanto questo testo verrà stampato sulla linea. Si tenga presente che gli spazi bianchi chiusi fra virgolette *non* sono ignorati, ma vengono stampati così come sono stati battuti. Ad esempio, se modifichiamo lo statement 110 del (3-29) in questo modo:

```
110   PRINT "      LE RISPOSTE SONO"                                (3-31)
```

la linea di testo risultante sarà

```
      LE RISPOSTE SONO
```

Virgole, nonché punti e virgola, possono usarsi anche per ottenere la spaziatura. Ad esempio,

```
150   PRINT "MEDIA","VOTO"
```

farà stampare

```
MEDIA          VOTO
```

Si osservi che, dopo che è stato stampato MEDIA, la testina di stampa, per effetto della virgola, si porterà all'inizio della zona successiva, a partire dal quale sarà stampato VOTO. Questa procedura è utile quando si vogliono stampare intestazioni di tabelle.

3-5-1. Stampa di dati con lo statement di INPUT

Nel paragrafo 3-3 abbiamo parlato dello statement di INPUT: quando questo viene eseguito, sulla telescrivente compare un ? per avvertire l'utilizzatore che devono essere introdotti dei dati. Ma a volte l'utilizzatore non è sicuro di quali siano i dati necessari: mediante lo statement di PRINT si potrebbe eliminare questo problema. Vediamo un esempio: supponiamo che si debbano introdurre tre voti, riportati in tre esercitazioni, che saranno associati alle variabili G1, G2 e G3. Consideriamo il seguente segmento di programma:

```
10   PRINT "INTRODURRE VOTO 1, VOTO 2, VOTO 3"
20   INPUT G1, G2, G3                                            (3-32)
```

Eseguendo il programma, verrà stampato quanto segue:

```
INTRODURRE VOTO 1, VOTO 2, VOTO 3
?                                                        (3-33)
```

Ora l'utilizzatore sa esattamente quali dati si devono introdurre.

Si tenga presente che il testo contenuto nello statement di PRINT non influisce sul resto del programma, in quanto una frase posta fra virgolette non modifica l'elaborazione.

Se lo statement di PRINT termina con una virgola, dopo l'esecuzione dello statement di PRINT la testina di stampa non tornerà ad inizio riga, e la carta non avanzerà: in questo caso il testo stampato si presenterà con una forma diversa, spesso preferibile. Ad esempio con

```
10  PRINT "A =",
20  INPUT A                                                  (3-34)
```

durante l'esecuzione del programma verrà stampato:

```
A =          ?
```

Anche in questo caso l'utilizzatore batterà il valore numerico di A dopo il punto interrogativo; se lo statement di PRINT termina con un punto e virgola, invece che con una virgola, la spaziatura sarà più stretta.

Vediamo un altro esempio su questo punto:

```
10  PRINT "VOTO 1, VOTO 2, E VOTO 3 =";
20  INPUT G1, G2, G3                                         (3-35)
```

Ne risulterà:

```
VOTO 1, VOTO 2, E VOTO 3 = ?                                 (3-36a)
```

L'utilizzatore allora introdurrà dopo il punto interrogativo tre numeri, separati da virgole. Una tipica linea di testo sarà ad esempio questa:

```
VOTO 1, VOTO 2, E VOTO 3 = ? 92,86,95                        (3-36b)
```

3-6. LO STATEMENT DI PRINTUSING

Parliamo in questo paragrafo di una tecnica avanzata di programmazione: il lettore può anche riservarsi di leggerlo quando avrà acquisito una maggiore esperienza.

Quando come risultato dello statement di PRINT vengono stampati dei numeri, il formato dell'uscita non può essere completamente specificato: ad esempio non può essere definito il numero di posizioni decimali. (Apporteremo delle parziali modifiche a questo punto nel paragrafo 3-8.) Oppure i dati devono essere collocati a partire dal margine sinistro della zona, e cosi via. In questo paragrafo tratteremo una procedura per mezzo della quale si possono specificare cose come la collocazione dei dati e le posizioni decimali. Gli statements di cui si parlerà qui non sono standard in tutte le versioni di BASIC, nel senso che le relative realizzazioni possono variare da una versione all'altra. Per di più le procedure descritte in questo paragrafo non sono assolutamente utilizzabili con alcune versioni: rinviamo perciò al manuale BASIC del calcolatore utilizzato la verifica di quali procedure, tra quelle di cui stiamo parlando, possano essere usate.

La parola-chiave del BASIC che ci permette di ottenere la maggiore versatilità che ci occorre è PRINTUSING. Questa parola-chiave è usata in modo analogo alla parola-chiave PRINT, per il fatto che contiene una lista di variabili. Vediamo un esempio:

```
110   PRINTUSING 120,A,B,X                                        (3-37)
```

La forma dello statement è: numero di statement, la parola-chiave PRINTUSING, un numero, una virgola, una lista di variabili separate da virgole. Si osservi che non c'è virgola fra PRINTUSING ed il numero, né alla fine della sequenza. Lo statement di PRINTUSING si usa insieme con un altro statement, detto *statement-immagine*, la cui funzione è di fornire il formato dei dati di stampa: il numero di statement dello statement-immagine è quello che compare nello statement di PRINTUSING. Ad esempio, nel segmento (3-37), il numero dello statement-immagine è 120. Vediamo un esempio:

```
110   PRINTUSING 120,A,B,X
120   :A = # # #. # # IL NUMERO DI LIBRI E' # # # E
       X = # # #. #                                               (3-38)
```

Se A = 261.368, B = 123, ed X = 27.4367, come risultato dell'esecuzione dello statement 120 verrà stampato:

```
A = 261.37 IL NUMERO DI LIBRI E' 123 E X = 27.4
```

Soffermiamoci sulle implicazioni di quanto si è detto. Lo statement-immagine, che

indica il formato di stampa, ha la forma seguente: numero di statement, due punti, e infine tutto ciò che definisce la stampa. Se vogliamo che venga stampato un testo, questo compare nello statement-immagine e, si badi bene, non fra virgolette. Per definire il formato di stampa dei dati numerici si usa il simbolo # : ciascun gruppo di tali simboli, che può avere o no un punto decimale al suo interno, indica che dev'essere stampato un numero. Fra gruppi di # possono essere intercalate parti di testo.

Riferendosi allo statement-immagine 120 per un maggior chiarimento, in primo luogo è stampato il testo A =. Il gruppo di # indica che devono essere stampati dei dati: di conseguenza è stampato il valore di A. Poi è stampato il testo IL NUMERO DI LIBRI È. A questo punto troviamo un altro gruppo di # , che determina la stampa della variabile successiva, che è B. Poi è stampato il testo E X =. Da ultimo, troviamo il terzo gruppo di # , che determina la stampa del valore di X.

Riassumiamo quello che si è detto: lo statement di PRINTUSING specifica le variabili che devono essere stampate (nel nostro esempio tre: A, B e X); lo statement-immagine specifica il testo che dev'essere stampato, nonché la collocazione ed il formato dei dati.

Per quanto concerne il formato di stampa dei dati numerici, si possono usare tre forme di gruppi di # . Esaminiamole una per volta.

3-6-1. Formato decimale, detto anche formato F

Qui abbiamo uno o più # con associato un punto decimale. Ad esempio, il gruppo # #. # # # fa sì che il calcolatore stampi il numero con due cifre a sinistra, e tre cifre a destra del punto decimale. Così, 27.13896 sarà stampato come 27.139, cioè sarà arrotondato a tre cifre dopo il punto decimale. Se il numero richiede più di due cifre alla destra del punto decimale, allora sorge un problema. Supponiamo ad esempio di cercare di stampare 127.13896 con la specifica precedente: a seconda della versione di BASIC, si avrà in stampa una delle tre seguenti possibilità:

```
127.139
27.139
MESSAGGIO DI ERRORE
```

Nel primo caso, il calcolatore ha prevalso sulla specifica del programma, che portava a stampare una risposta errata. Nel secondo caso, la specifica è stata seguita esattamente, ed è stato stampato un dato sbagliato: infatti il programma specificava due sole cifre a sinistra della virgola, e due ne sono state stampate, per quanto la risposta sia del tutto errata. (Usando questa specifica, il programmatore deve stare molto attento, per essere sicuro che alla sinistra del punto decimale venga fornito un numero sufficiente di cifre.) Nel terzo caso, se non viene fornito un numero sufficiente di cifre alla sinistra del punto decimale, o risulteranno dei dati errati, oppure sarà stampato un messaggio di errore. Dalla consultazione del manuale di BASIC

che correda il calcolatore usato potrete vedere quale tipo di risposta dovete aspettarvi: si noti che, su questo punto, compilatori differenti danno risultati molto diversi.

Con alcune (non tutte) versioni di BASIC si può verificare ancora un'altra situazione. Ad esempio, se vi è almeno un # alla sinistra del punto decimale, allora non si terrà conto della specifica, se necessario. Ad esempio, se è specificato #.# # #, 27.13645 sarà stampato così: 27.136. Se al contrario non è dato nessun # alla sinistra del punto decimale, si tiene conto delle specifiche. Ad esempio, se è specificato .# # #, 27.13645 verrà stampato così: .136, il che è completamente sbagliato. Quindi uno stesso compilatore può in un caso non tener conto della specifica e in un altro caso tenerne conto.

Su questo punto si possono avere molte soluzioni diverse, a seconda della versione di BASIC: il vostro manuale di BASIC vi dirà come si comporterà il calcolatore.

Quando è utilizzato questo tipo di specifica, non si ricorre alla notazione con E per rappresentare dei numeri. Così, se è specificato # #. # # #, ed il numero da stampare è 1.2E8, sarà stampato, a seconda del compilatore, 1200000000, o 00.000, o un messaggio di errore. Si tenga presente che i compilatori BASIC non sono standard per quanto riguarda lo statement di PRINTUSING, e che è possibile che il vostro compilatore vi dia un altro risultato ancora, diverso dai tre appena illustrati.

3-6-2. Formato numero intero, detto anche formato I

Volendo stampare soltanto un numero intero, la forma è essenzialmente la stessa di quella del formato decimale, tranne che è omesso il punto decimale. Ad esempio, # # # determinerà la stampa di un numero di tre cifre al massimo. Se il numero ha meno di tre cifre, compariranno degli spazi alla sua sinistra. Se ad esempio abbiamo

```
150  :PRINTUSING 160,A,B
160  :LO STUDENTE NUMERO # # # HA PRESO # # # # LIBRI    (3-39)
```

per A = 15 e B = 2, il testo che verrà stampato è

```
LO STUDENTE NUMERO  15 HA PRESO   2 LIBRI                (3-40)
```

Se, usando questo formato, stampiamo numeri non interi, la parte frazionaria sarà *troncata.* Così, nell'esempio precedente, se avessimo A = 15.967 e B = 2.03, il testo stampato sarebbe esattamente quello di (3-40), in quanto A sarebbe troncato a 15 e B a 2. Si osservi che nell'uscita non compare il punto decimale, e che il troncamento *non* è un arrotondamento: 15.967 è troncato a 15, *non* arrotondato a 16.

Se il numero da stampare comprende più cifre di quelle specificate dai segni #, come prima i risultati saranno diversi da una versione di BASIC all'altra: o il risultato sarà stampato correttamente, cioè come se in partenza fossero stati specificati de-

gli ulteriori #, o si avranno dei dati errati, o verrà stampato un messaggio di errore. Il concetto di base essenzialmente è lo stesso che per il formato decimale.

3-6-3. Formato esponenziale, detto anche formato E

Possiamo specificare che il numero sia stampato ricorrendo alla notazione con E, facendo seguire ai segni # quattro punti esclamativi, ! (né più né meno). Ad esempio,

```
#. # # #!!!!
```

Ci devono essere *sempre* almeno un # alla sinistra del punto decimale ed esattamente quattro ! alla fine della specifica. I !!!! specificano quattro spazi, destinati, nell'ordine, a: una E, un segno, un esponente di due cifre. Se abbiamo ad esempio

```
190   PRINTUSING 200,A
200   :A È UGUALE A #. # # #!!!!                    (3-41)
```

e se il valore di A è 26.137892, all'esecuzione dello statement 190 sarà stampato quanto segue:

```
A E' UGUALE A 2.614E+01                             (3-42)
```

Ma sono possibili delle varianti: ad esempio, se l'esponente è positivo, il segno + che vien dopo la E può essere omesso.

Anche qui raccomandiamo al lettore di tener conto delle diversità che si presentano da una versione di BASIC all'altra; ad esempio alcuni compilatori ammettono l'uso delle specifiche di numero decimale e di numero intero, ma non di quella esponenziale.

3-6-4. Uso dei segni con gli specificatori di formato

Con alcuni compilatori può esserci un segno accanto allo specificatore #. Si danno tre possibilità: se un segno + precede lo specificatore (ad esempio, + # #. # # #), verrà stampato un segno + prima dei dati se il numero da stampare è positivo, mentre, se il numero è negativo, questo sarà stampato preceduto dal segno meno. Se lo specificatore è preceduto da un segno meno (ad esempio, – # #. # # #), il numero da stampare sarà preceduto dal segno meno solo se è negativo. Se nessun segno precede lo specificatore (ad esempio, # #. # # #), il numero, se positivo, è stampato senza segno, se negativo, è preceduto dal segno meno, che prende il po-

sto di uno dei segni #: ad esempio, se è specificato # #. # # #, sarà stampato un numero negativo, come se si fosse specificato – #. # # #.

Ad ogni buon conto, con alcuni compilatori non devono essere usati i segni, dato che questi verrebbero interpretati come testo, e genererebbero una stampa errata. Ad esempio, – # #. # # # farebbe *comunque* stampare il segno meno (perché il segno meno è interpretato come testo), e quindi un numero positivo sarebbe preceduto dal segno meno, ed un numero negativo da due segni meno.

Se lo statement di PRINTUSING contiene un numero di variabili maggiore degli specificatori di formato contenuti nello statement-immagine ad esso associato, lo statement-immagine sarà riutilizzato ed ogni volta la stampa riprenderà su una nuova linea. Vediamo un esempio:

```
100   PRINTUSING 120,A,B,C
120   :RISP = # #. # #                                   (3-43)
```

Se A = 26.137, B = 46.236 e C = 19.217, si avrà in uscita:

```
RISP = 26.14
RISP = 46.24
RISP = 19.22                                             (3-44)
```

Abbiamo illustrato lo statement di PRINTUSING come contenente una lista di variabili; in realtà, esso può contenere espressioni aritmetiche, che saranno in tal caso calcolate e stampate come numeri singoli. Il concetto di base è lo stesso di quello illustrato a proposito dello statement di PRINT (v. par. 3-4).

3-7. LO SPECIFICATORE TAB

Si può variare la spaziatura dei dati inserendo lo specificatore TAB (tabulatore) nello statement di PRINT. Vediamo un esempio:

```
110   PRINT A; TAB(17), B; TAB(38), C                    (3-45)
```

All'esecuzione di questo statement, A verrà stampata a partire dalla colonna 1, B a partire dalla colonna 17, e C a partire dalla colonna 38. Fra le parentesi dello specificatore TAB può comparire una variabile o un'espressione. Ad esempio, questa sequenza è equivalente alla (3-45):

```
 90   LET M = 17.6
```

```
100  LET N = 2.2
110  PRINT A; TAB(M), B; TAB(M*N), C                        (3-46)
```

Questo perché, se il valore numerico posto fra parentesi non è un numero intero, viene troncato, ovvero il punto decimale è omesso: così 17.6 è troncato a 17, e allo stesso modo M*N = 38.72 è troncato a 38. Pertanto gli statements 110 dei segmenti di programma (3-45) e (3-46) sono equivalenti.

Dopo gli specificatori TAB compaiono delle virgole; in alcune versioni di BASIC, però, si può usare il punto e virgola. Sono possibili anche altre varianti: per questo si consulti il manuale di BASIC del calcolatore. Analogamente il punto e virgola che segue la variabile può essere sostituito da una virgola: in entrambi i casi la loro interpretazione è la stessa che in un normale statement di PRINT. Analizziamo questo punto. Supponiamo di avere

```
120  PRINT A, TAB(17), B, TAB(36), C                        (3-47)
```

All'esecuzione di questo statement, A sarà stampata a partire dalla colonna 1. La virgola farà avanzare la testina di stampa fino alla colonna 16, il TAB(17) la porterà alla colonna 17, e a questo punto sarà stampata B. La virgola che segue B porterà la testina di stampa alla colonna 31, quindi il TAB(36) la porterà alla colonna 36, dove verrà stampata C.

In uno statement di PRINT possiamo collocare un TAB prima della prima variabile, se non vogliamo che il primo numero venga stampato a partire dalla colonna 1.

Lo specificatore TAB non può essere usato per riportare indietro la testina di stampa: in questi casi sarà ignorato. Consideriamo lo statement

```
120  PRINT A, TAB(12), B, TAB(60), C                        (3-48)
```

A sarà stampata a partire dalla colonna 1. La virgola farà avanzare la testina di stampa fino alla colonna 16, per cui il TAB(12) sarà ignorato: se il TAB è ignorato, la virgola seguente può o no (a seconda della versione di BASIC) far avanzare la testina di stampa fino alla colonna 31. Quindi B sarà stampata a partire dalla colonna 31 (o 16). La virgola che segue B farà avanzare poi la testina di stampa fino alla colonna 46 (o 31); pi il TAB(60) porterà la testina di stampa alla colonna 60 e, a partire da questa colonna, sarà stampata C.

Usando punti e virgola invece di virgole dopo le variabili, si ottiene una spaziatura più stretta, che riduce al minimo i salti dovuti agli specificatori TAB. (Quindi, in linea generale, conviene usare la forma dello statement (3-45) invece di quella del (3-47).) Si tenga presente che anche la lunghezza dei dati può portare ad ignorare lo specificatore TAB. Consideriamo ad esempio

```
120  PRINT A, TAB(28), B, TAB(39), C
```

B sarà stampata a partire dalla colonna 28. Se è formata da più di tre cifre, l'ultima cifra cadrà nella colonna 31, o oltre: in questo caso, la virgola che segue B porterà la testina di stampa alla colonna 46, ed il TAB(39) sarà ignorato. Se invece B contiene due sole cifre, la seconda verrà stampata nella colonna 30, e la virgola che segue B porterà la testina di stampa alla colonna 31: in questo caso il TAB(39) non è ignorato.

3-8. LO STATEMENT DI SETDIGITS

In certi casi, si desidera poter specificare il numero di cifre che vengono stampate senza ricorrere allo statement di PRINTUSING: lo si può fare con lo statement di SETDIGITS. Consideriamo l'esempio seguente:

```
35   SETDIGITS(3)                                          (3-49)
```

Allora tutti gli statements di PRINT che *seguono* questo statement stamperanno i loro dati con tre sole cifre significative e, se è necessario, si userà la notazione con E. Il termine posto fra parentesi non dev'essere necessariamente un numero intero; può essere anche una variabile o un'espressione. Se i valori numerici della variabile o dell'espressione non sono interi, saranno troncati (cioè la parte frazionaria sarà soppressa (v. par. 3-7)). Consideriamo l'esempio seguente:

```
 10  LET A = 263.67894
 20  LET B = 2.5
 30  PRINT A
 40  SETDIGITS(1)
 50  PRINT A
 60  SETDIGITS(B)
 70  PRINT A
 80  SETDIGITS(2*B)
 90  PRINT A
100  END                                                   (3-50)
```

L'esecuzione di questo programma porterà alla stampa di

```
263.67894
3E+02
2.6E+02
263.68
```

Lo statement 30 fa sì che A venga stampato con otto cifre significative, che noi supponiamo essere la notazione BASIC standard (che può variare da calcolatore a calcolatore). Lo statement 40 impone che la stampa successiva avvenga con una cifra significativa. Allora, all'esecuzione dello statement 50, abbiamo 3E+02, che equivale a 300, e rappresenta l'arrotondamento di 263.67894 ad una cifra significativa: questo quindi richiede la notazione con E, che è infatti usata. Il valore di B è 2.5, che, troncato, diventa 2. Quindi lo statement 60 è equivalente a SETDIGITS(2). All'esecuzione dello statement 70, si ottiene 2.6E+0 = 260. Lo statement 80 è equivalente a SETDIGITS(5). Pertanto l'esecuzione dello statement 90 ha come risultato 263.68. La notazione con E in questo caso non è richiesta, e perciò non è usata.

Si tenga presente che non tutte le versioni di BASIC consentono l'uso dello statement di SETDIGITS. Dal momento che SET è una parola-chiave del BASIC (ne parleremo nel paragrafo 11-1), usando lo statement di SETDIGITS con un compilatore che non lo ammette potranno comparire degli strani messaggi di errore.

ESERCIZI

3-1. Qui di seguito sono trascritti tutti gli statements di READ e di DATA di un programma in BASIC. Determinate i valori assegnati a tutte le variabili.

```
10 READ A,B,C
20 READ D
30 READ E,F,G
 .
 .
 .
100 DATA 10.7, 1E6, 2.3E-03, 1.45
110 DATA 22.6, 15.3, 19.7, 26.413
```

3-2. Esponete il concetto di blocco di dati.

3-3. Ripetete l'Esercizio 3-1 con

```
10 READ A,B,C,D
20 READ E,F,G
30 RESTORE
40 READ H,I,J,K
 .
 .
 .
100 DATA 1.2, 3.67, 1E5, 2.3E6, 1.678, 2.357
110 DATA 1.438, 2.196, 7.132, 9.367
```

3-4. Esponete il concetto di puntatore di dati.

3-5. Determinate i valori di X e di Y che risultano dall'esecuzione del programma seguente. Verificate il risultato ottenuto eseguendo il programma su di un calcolatore.

```
 10  READ A,B
 20  LET A=A+B↑2
 30  LET B=A*B
 40  RESTORE
 50  PRINT A,B
 60  READ A
 70  READ A,B
 80  LET A=A + 2*B
 90  LET B=A + 3
100  DATA 2,3,4
110  END
```

3-6. Descrivete l'azione dello statement seguente quando il programma in cui esso compare è eseguito.

```
10  INPUT A,B,C
```

3-7. Ripetete l'Esercizio 3-6 con la sequenza di programma:

```
10  INPUT A,B,C
20  LET D=A*B*C
30  INPUT X1,X2
```

3-8. Scrivete un programma in BASIC che contenga degli statements di INPUT ed eseguitelo su di un calcolatore. Il programma deve accettare tre numeri e ricavarne la media.

3-9. Dite quale uscita si avrà in seguito all'esecuzione dello statement seguente:

```
70  PRINT X,Y,A,B,C
```

in cui i valori assegnati alle variabili sono: A = 10.76, B = –23.2E4, C = 21.673, X = 15.967 ed Y = 3.2164. Verificate i risultati ottenuti scrivendo un appropriato programma in BASIC ed eseguendolo su di un calcolatore.

3-10. Ripetete l'Esercizio 3-9 con lo statement seguente:

```
80  PRINT X,Y,A,B,C,D,E
```

dove i valori assegnati sono gli stessi che nell'Esercizio 3-9 e, in più, D = 21.43678 ed E = 2.16913E6.

3-11. Ripetete l'Esercizio 3-9, ma questa volta con il seguente statement:

```
70  PRINT X;Y;A;B;C
```

3-12. Ripetete l'Esercizio 3-10, ma questa volta con il seguente statement:

```
80   PRINT X;Y;A;B;C;D;E
```

3-13. Esponete il concetto di zona in relazione agli statements di PRINT.

3-14. Ripetete l'Esercizio 3-9, ma con i seguenti statements:

```
70   PRINT X,Y,A
75   PRINT B,C
```

3-15. Ripetete l'Esercizio 3-14, ma con in più una virgola dopo la A nella linea 70.

3-16. Ripetete l'Esercizio 3-9, ma questa volta con il seguente statement di PRINT:

```
70   PRINT X,Y,A + B↑2,A*(X+Y),2*B
```

3-17. Dite quale uscita si avrà sulla telescrivente eseguendo lo statement seguente:

```
150 PRINT "VELOCITA'" = ",A,"TEMPO" =",B
```

dove i valori assegnati ad A e a B sono rispettivamente 22.6 e 15.4. Verificate i risultati ottenuti scrivendo un appropriato programma in BASIC ed eseguendolo su di un calcolatore.

3-18. Ripetete l'Esercizio 3-17, ma mettendo dei punti e virgola al posto delle virgole.

3-19. Un oggetto che si muova, a partire dalla condizione di riposo, con un'accelerazione a, dopo t secondi ha una velocità finale di

$$v = at$$

ed ha percorso una distanza s uguale a

$$s = 1/2at^2$$

Scrivete un programma in BASIC che acquisisca a e t da telescrivente durante l'esecuzione, e stampi v ed s, con opportune parole d'identificazione, come VELOCITA'=, etc. Verificate il programma eseguendolo su di un calcolatore.

3-20. Ripetete l'Esercizio 3-19, ma questa volta facendo in modo che sulla telescrivente compaia il testo:

```
INTRODURRE ACC E TEMPO
```

prima dell'introduzione dei dati, che dovranno comparire su una linea a parte.

3-21. Ripetete l'Esercizio 3-20, ma questa volta i dati compariranno sulla stessa linea del testo di guida.

3-22. Dite quale uscita si avrà eseguendo lo statement seguente:

```
150   PRINTUSING 160,A,B,C
160   :A = # #. # # # IL TASSO E' # #. #, IL NUM E' # # #
```

Sappiamo che i valori di A, B e C sono già stati assegnati come segue: A = 94.36789, B = 72.3176, C = 14. Verificate i risultati ottenuti scrivendo un programma opportuno ed eseguendolo su di un calcolatore.

3-23. Ripetete l'Esercizio 3-22, ma con le seguenti assegnazioni: A = 156.2, B = 716.34876, C = 143.2.

3-24. Ripetete l'Esercizio 3-22, essendo però lo statement - immagine

```
160   : A = #. # # #!!!! IL TASSO E' # # #, IL NUM E' # #. #
```

3-25. Illustrate l'uso dei segni con gli specificatori di formato nello statement-immagine.

3-26. Dite quale uscita si avrà eseguendo il seguente statement:

```
100   PRINT A;TAB(14),B;TAB(J),C;TAB(2*X)
```

Supponiamo che i valori assegnati alle variabili siano: A = 1, B = 2, C = 3, J = 28, X = 25. Verificate i risultati ottenuti scrivendo un opportuno programma in BASIC ed eseguendolo su di un calcolatore.

3-27. Ripetete l'Esercizio 3-26, ma con virgole al posto dei punti e virgola.

3-28. Ripetete l'Esercizio 3-9, ma con, in più, lo statement seguente:

```
60   SETDIGITS(D)
```

dove a D è già stato assegnato il valore 3.

3-29. Ripetete l'Esercizio 3-28, ma questa volta assumendo 1 come valore di D.

CAPITOLO 4

GLI STATEMENTS DI CONTROLLO

In tutti i programmi in BASIC esaminati finora gli statements erano eseguiti nell'ordine. Vedremo ora degli statements che portano il calcolatore a seguire varie ramificazioni del programma, e spiegheremo come questo possa essere messo in rapporto con un procedimento mediante il quale vengono prese decisioni, il che spesso può essere molto utile.

Gli statements di cui parleremo sono detti *statements di controllo*; essi si basano su concetti che sono diversi da quelli esaminati finora.

Illustreremo anche un procedimento grafico di rappresentazione di un programma, che si rivela spesso utile per chi scrive i programmi, ed anche per chi legge programmi scritti da altri. Parleremo poi anche di altri procedimenti usati per descrivere che cosa fa un programma, anch'essi utili sia a chi scrive i programmi che a chi li legge.

4-1. LO STATEMENT DI GO TO

Le operazioni del calcolatore sono controllate dagli statements BASIC del programma. Fino ad ora siamo partiti dal presupposto che questo controllo passasse da uno statement al successivo nell'ordine sequenziale. Esamineremo adesso una procedura per mezzo della quale quest'ordine può venire modificato, ed il controllo può essere trasferito a qualunque statement si desideri. Uno statement BASIC che assolve a questa funzione è lo statement di GO TO, che ha la forma

```
90   GO TO 30
```
(4-1)

Quando lo statement 90 è eseguito, il controllo sarà trasferito allo statement numero 30. Comunque, prima di descriverne l'azione, vediamo la forma dello statement di GO TO: esso consiste del numero di statement e della parola-chiave GO TO,

seguita da un numero, che *deve* corrispondere ad un numero di statement presente nel programma in BASIC.

Illustriamo l'uso dello statement di GO TO con un programma semplice da eseguirsi in timesharing. Supponiamo di voler calcolare la media di gruppi di tre numeri, e di voler inoltre ripetere questo calcolo più volte. Potremmo scrivere un programma che calcoli la media di tre numeri ed eseguirlo più volte, ma sarebbe un metodo lento. Ad esempio (v. par. 1-4), ad ogni esecuzione del programma, si dovrebbe battere la parola RUN, ed il calcolatore dovrebbe rispondere con l'intestazione. È uno spreco di tempo. Vediamo ora il programma seguente, che elimina questo problema:

```
10  REM PROGRAMMA ILLUSTRANTE
15  REM LO STATEMENT DI GO TO
20  PRINT "INTRODURRE A,B,C"
30  INPUT A,B,C
40  LET D = (A+B+C)/3
50  PRINT "MEDIA =";D
60  GO TO 20
70  END                                          (4-2)
```

Esaminiamo lo svolgimento di questo programma. Quando parte l'esecuzione, sarà stampato INTRODURRE A,B,C, poi gli statement 30-50 provvedono all'ingresso dei dati, al calcolo della media ed alla stampa del risultato. A questo punto è eseguito lo statement 60, ed il controllo ritorna allo statement 20: viene stampato di nuovo INTRODURRE A,B,C, e le operazioni precedenti vengono ripetute. Un'esecuzione tipica si presenterà così:

```
>RUN
MED          12:30          15-FEB-77
INTRODURRE A,B,C
? 50,100,80
MEDIA = 76.66667
INTRODURRE A,B,C
? 80,70,90
MEDIA = 80

                                             (4-3)
```

Questo svolgimento si ripeterà ciclicamente per un numero illimitato di volte, senza fermarsi. È comunque possibile inviare un segnale di interruzione al calcolatore per fermare l'elaborazione: per questa funzionalità consultate il manuale di BASIC

del calcolatore utilizzato. Allora, eseguito il calcolo un certo numero di volte, inviando il segnale d'interruzione dell'operazione l'elaborazione ha termine.

Una variante del programma (4-2) è:

```
10  REM PROGRAMMA CHE SI FERMA
15  REM AUTOMATICAMENTE
20  READ A,B,C
30  LET D = (A+B+C)/3
40  PRINT "MED =";D
50  DATA 50,100,80,80,70,90
60  GO TO 20
70  END                                                    (4-4)
```

Ora, all'esecuzione del programma, si ha:

```
>RUN
MED            12:45            15-FEB-77
MED = 76.66667
MED = 80
% OUT OF DATA IN 20
>                                                          (4-5)
```

Lo statement 20, insieme con lo statement di DATA, determina le assegnazioni: A = 50, B = 100, C = 80: allora, dopo l'esecuzione dello statement di READ, il puntatore si trova posizionato sul quarto elemento del blocco di dati. Quindi, dopo che gli statements 30 e 40 sono eseguiti, viene stampata la media 76.66667. A questo punto lo statement di GO TO fa tornare il controllo allo statement di READ numero 20. Essendo il puntatore sul quarto elemento del blocco di dati, verranno fatte le seguenti assegnazioni: A = 80, B = 70, C = 90; quindi la media 80 viene calcolata e stampata. Il controllo torna poi allo statement di READ numero 20, ma a questo punto, non essendoci un numero sufficiente di dati, è generato il messaggio OUT OF DATA e l'elaborazione s'interrompe. Se fossero stati forniti più dati, il programma avrebbe continuato fino al loro esaurimento: in conclusione, il programma è tale da fermarsi da solo automaticamente.

4-2. LO STATEMENT DI IF-THEN

Lo statement di GO TO illustrato nell'ultimo paragrafo è definito statement *di salto non condizionato*. La parola "salto" indica che il controllo passa ad uno statement diverso da quello immediatamente successivo; l'espressione "non condizionato" in-

dica che il salto è sempre il medesimo, indipendentemente dai valori delle variabili. Tratteremo ora del più potente statement *di salto condizionato*, nel quale la diramazione dipende dai valori delle variabili. Lo statement di salto condizionato permette al programmatore di prendere decisioni nel programma.

Lo statement che si usa per effettuare il salto condizionato è detto statement di IF-THEN. Vediamolo:

```
50  IF A = 50 THEN 120                                              (4-6)
```

Esaminiamo la forma di questo statement: numero di statement, la parola-chiave IF, una *relazione* (nel nostro esempio A = 50), seguita dalla parola THEN e da un numero, che *deve* essere un numero di statement presente nel programma in BASIC. L'azione dello statement IF-THEN è la seguente: se la relazione è vera (nel nostro esempio, se A è davvero uguale a 50), il controllo passa immediatamente allo statement 120. Se la relazione non è vera (nel nostro esempio, se A non è uguale a 50), il controllo passa al primo statement che vien dopo lo statement di IF-THEN (di fatto, lo statement di IF-THEN è ignorato). Consideriamo ad esempio la sequenza

```
 50  IF A = 35 THEN 90
 60  B = 2*A
 80  GO TO 100
 90  B = 3*A
100  PRINT B
```

(4-7)

Analizziamo quello che fa il programma. Se A = 35, quando lo statement 50 è eseguito il controllo passa allo statement 90. Poi viene calcolato B = 3*A, e quindi il controllo passa allo statement successivo, che è lo statement 100, con cui viene stampato B. Quindi il programma prosegue. Se invece A non è uguale a 35, lo statement di IF-THEN viene praticamente ignorato. Il controllo allora passa allo statement 60 e si calcola B = 2*A. Viene ora eseguito lo statement successivo, lo statement 80, che trasferisce il controllo allo statement 100, con cui è stampato B. Quindi il programma prosegue. In conclusione, si osservi che il controllo procede sempre da uno statement a quello successivo, a meno che trovi uno statement di salto.

In questo esempio, la relazione presente nello statement di IF-THEN era un'uguaglianza, ma non è indispensabile che sia sempre così. Ad esempio, potremmo avere

```
90  IF X1 > 50 THEN 30
```

In questo caso, se X1 è maggiore di 50, il controllo passa allo statement 30; se X1 non è maggiore di 50, lo statement di IF-THEN è ignorato.

Negli statements di IF-THEN si usano diversi *operatori di relazione*. Ne abbiamo visti due: uguale (=) e maggiore di (>). In Tabella 4-1 diamo un elenco di tutti quelli che si usano.

=	Uguale a
>	Maggiore di
<	Minore di
>=	Maggiore o uguale a
<=	Minore o uguale a
< >	Diverso da

Tabella 4.1 — Operatori di relazione.

Illustriamo con un programma l'uso dello statement di IF-THEN. Supponiamo di scrivere un programma che calcoli la media di tre voti (introdotti da tastiera), e quindi associ una lettera-voto alla media. La lettera-voto viene determinata in base allo schema seguente:

A Media uguale o maggiore di 90
B Media uguale o maggiore di 80 ma minore di 90
C Media uguale o maggiore di 70 ma minore di 80
D Media uguale o maggiore di 60 ma minore di 70
F Media minore di 60

Un programma che esegue queste operazioni è il seguente:

```
 10  REM PROGRAMMA DI CLASSIFICAZIONE
 20  PRINT "INTRODURRE PROVA 1, PROVA 2, PROVA 3"
 30  INPUT T1, T2, T3
 40  LET A = (T1+T2+T3)/3
 50  IF A < 90 THEN 80
 60  PRINT "IL VOTO E' A"
 70  GO TO 999
 80  IF A < 80 THEN 110
 90  PRINT "IL VOTO E' B"
100  GO TO 999
110  IF A < 70 THEN 140
120  PRINT "IL VOTO E' C"
```

```
130  GO TO 999
140  IF A < 60 THEN 170
150  PRINT "IL VOTO E' D"
160  GO TO 999
170  PRINT "IL VOTO E' F"
999  END
```
(4-8)

Esaminiamo quello che fa il programma. Gli statements 20-40 fanno sì che i tre voti siano introdotti da tastiera e che ne sia calcolata la media. Consideriamo ora lo statement 50. Se la media è minore di 90, il controllo passa allo statement 80; se invece la media A è uguale o maggiore di 90, lo statement 50 è ignorato. In questo caso viene eseguito lo statement 60 ed è stampato

IL VOTO E' A

Poi è eseguito lo statement 70, che trasferisce il controllo allo statement 999, lo statement di END, e l'elaborazione termina.

Se A è minore di 90, il controllo passa allo statement 80 e a questo punto, se A è minore di 80, il controllo passa allo statement 110. Se invece A è uguale o maggiore di 80, il controllo passa allo statement 90. In questo caso, verrà stampato

IL VOTO E' B

Poi sarà eseguito lo statement 100, che passa il controllo allo statement 999, e l'elaborazione si conclude. Si osservi che lo statement 80 verifica solo se A è minore di 80, poi, se A è uguale o maggiore di 80, è eseguito lo statement 90. Potrebbe sembrare da questo che il messaggio IL VOTO E' B verrebbe stampato anche se la media fosse maggiore di 90. In realtà non è così: essendo A uguale o maggiore di 90, gli statements 50 e 70 farebbero passare il controllo allo statement 999 prima che lo statement 80 venisse eseguito.

L'esecuzione del resto del programma è analoga, per cui alla fine saranno forniti i risultati appropriati. Nel paragrafo 4-4 vedremo come i programmi possono essere rappresentati graficamente allo scopo di facilitare la loro comprensione, e faremo anche altri esempi di salti condizionati.

Nello statement di IF-THEN deve comparire soltanto *una singola variabile* alla *sinistra* dell'operatore di relazione; alla destra dell'operatore di relazione invece può comparire sia un numero, sia una variabile, sia un'espressione. Ad esempio,

```
110  IF A > = B1 THEN 60
```
(4-9)

Supponiamo che i valori di A e di B1 siano stati assegnati in un punto precedente del programma. Allora, se il valore di A è uguale o maggiore del valore di B1, il pri-

mo statement ad essere eseguito sarà lo statement 60; in caso contrario il controllo passerà allo statement che segue allo statement 110.

Un altro statement di IF-THEN corretto è

```
130   IF A > B*A/2 THEN 40                                    (4-10)
```

Supponiamo che i valori di A e di B siano stati assegnati prima dello statement 130. Ora, se A è maggiore di B*A/2, sarà eseguito lo statement 40: se invece A è uguale o minore di B*A/2, sarà eseguito lo statement che vien dopo lo statement 130.

4-2-1. Gli statements di IF-GO TO

Alcune versioni di BASIC permettono che negli statements di salto condizionato alla parola THEN si sostituisca GO TO. Ad esempio, lo statement (4-10) si potrebbe scrivere

```
130   IF A > B*A/2 GO TO 40                                   (4-11)
```

Gli statements (4-10) e (4-11) sono identici. Si noti che, mentre tutti i calcolatori in BASIC lavorano con la forma IF-THEN, solo alcuni ammettono la forma IF-GO TO.

4-3. LO STATEMENT DI STOP

Uno statement BASIC che si rivela spesso utile è lo statement di STOP. La sua forma è:

```
90   STOP                                                     (4-12)
```

cioè numero di linea seguito dalla parola-chiave STOP. La sua funzione è quella di fermare l'elaborazione. Pertanto, durante l'esecuzione, STOP ed END hanno la stessa funzione, ma con delle differenze: END si usa anche per porre fine alla compilazione. Infatti in un programma può esserci un solo statement di END, che sarà necessariamente l'ultimo; gli statements di STOP invece possono essere diversi, e collocati in qualunque punto del programma.

Lo statement di STOP si rivela vantaggioso quando in un programma ci sono diversi rami, e si vuol porre termine all'elaborazione alla fine di ciascun ramo. Consideriamo ad esempio il programma (4-8), che contiene cinque rami, corrispondenti ciascuno ad uno dei voti A, B, C, D, F; quattro di questi rami finiscono con lo state-

ment GO TO 999, che dà il controllo allo statement di END e pone fine all'elaborazione. Una forma alternativa del programma è

```
 10  REM PROGRAMMA CHE USA
 15  REM STATEMENTS DI STOP
 20  PRINT "INTRODURRE PROVA 1, PROVA 2, PROVA 3"
 30  INPUT T1, T2, T3
 40  LET A = (T1+T2+T3)/3
 50  IF A < 90 THEN 80
 60  PRINT "IL VOTO E' A"
 70  STOP
 80  IF A < 80 THEN 110
 90  PRINT "IL VOTO E' B"
100  STOP
110  IF A < 70 THEN 140
120  PRINT "IL VOTO E' C"
130  STOP
140  IF A < 60 THEN 170
150  PRINT "IL VOTO E' D"
160  STOP
170  PRINT "IL VOTO E' F"
180  END                                          (4-13)
```

Questo programma è lo stesso dell'esempio (4-8), tranne che gli statements di GO TO sono stati sostituiti con degli statements di STOP. Dal punto di vista operativo, l'esecuzione è alquanto più semplice, perché non dev'essere eseguito lo statement di GO TO.

Inoltre, quando è usato lo statement di STOP, è anche più facile per il programmatore, leggendo un programma, localizzare la fine del ramo, perché in un programma possono esserci molti statements di GO TO che non indirizzano il controllo allo statement di END.

4-4. I DIAGRAMMI DI FLUSSO

Quando si usano degli statements di controllo, i programmi si fanno più complessi. Per poter scrivere dei programmi complessi, è necessario individuare l'appropriata strategia da usare, cioè una serie di stratagemmi che sono detti *algoritmi*. Vediamo l'algoritmo usato nei programmi (4-8) e (4-13). In questi programmi abbiamo calcolato la media di tre voti, ed abbiamo poi associato ad ogni media le lettere-voto A, B, C, D o F. L'algoritmo usato qui è stato:

1) Lettura dei tre voti T1, T2 e T3.

2) Calcolo della media A = (T1+T2+T3)/3.

3) Si verifica se A < 90. Se non lo è, si stampa A come voto e STOP.

4) Si verifica se A < 80. Se non lo è, si stampa B come voto e STOP.

5) Si verifica se A < 70. Se non lo è, si stampa C come voto e STOP.

6) Si verifica se A < 60. Se non lo è, si stampa D come voto e STOP.

7) Si stampa F come voto e STOP.
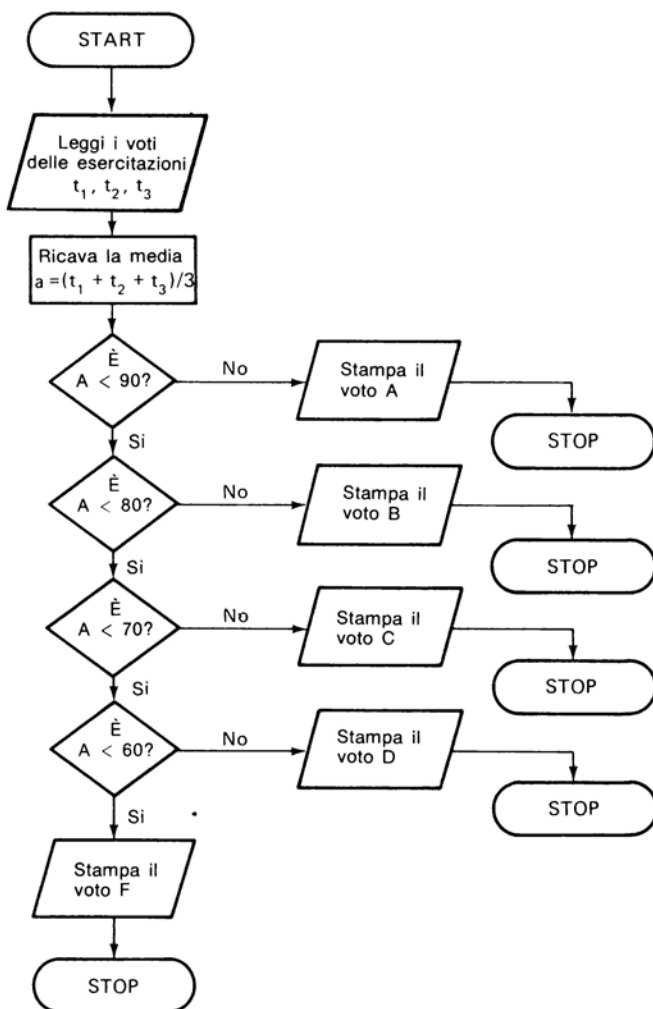
Figura 4.1 — Diagramma di flusso dell'algoritmo per il calcolo dei voti.

È spesso utile avere la rappresentazione grafica di un algoritmo: questa rappresentazione è detta *diagramma di flusso.* In Figura 4-1 si può vedere il diagramma di flusso dell'algoritmo precedente. Si osservi che ciascuno dei passi dell'algoritmo è rappresentato in un riquadro del diagramma di flusso. Come si può vedere, il diagramma di flusso aiuta a visualizzare l'algoritmo. La visualizzazione aiuta spesso sia ad organizzare i nostri pensieri in fase di esame dell'algoritmo, sia a seguire il corso dell'algoritmo stesso; da ultimo aiuta coloro che leggono un programma scritto da altri.
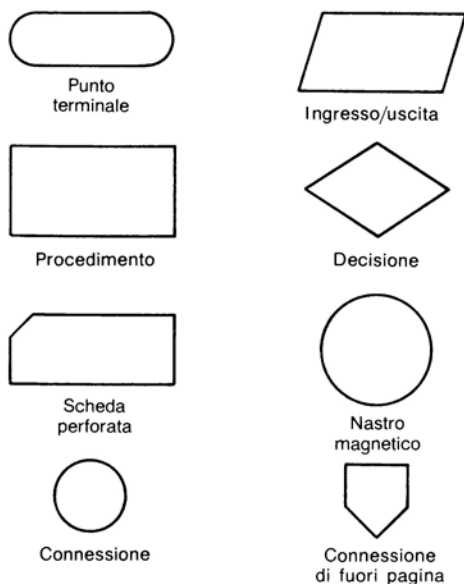
Figura 4.2 — Alcuni simboli dei diagrammi di flusso.

Nel diagramma di flusso compaiono riquadri di forma diversa, ciascuno dei quali serve a rappresentare un diverso tipo di operazione. In Figura 4-2 si trovano quelli usati comunemente. Esaminiamoli. Il simbolo di *punto terminale* indica i punti d'inizio e di fine. Il simbolo di *ingresso/uscita* rappresenta l'ingresso o l'uscita dei dati. Il simbolo di *processo* tipicamente rappresenta operazioni aritmetiche. Il simbolo di *decisione* rappresenta tipicamente il tipo di operazione IF-THEN, con cui si prende una decisione: da un simbolo di decisione si dipartono diverse ramificazioni, che saranno contrassegnate in modo da sapere quale diramazione bisogna prendere. Esaminiamo ad esempio, in Figura 4-1, il primo riquadro di decisione, partendo dall'alto. In esso sono indicate la decisione da prendere (se $A < 90$), e le diramazioni che si dipartono (contrassegnate con un sì ed un no): se A è realmente minore di 90, l'operazione prosegue lungo la diramazione "sì"; se invece A non è minore di 90, l'opera-

zione prosegue lungo la diramazione "no". I simboli di *scheda perforata* e di *nastro magnetico* rappresentano dispositivi particolari d'ingresso/uscita.

I simboli del diagramma di flusso sono collegati da linee terminanti con una freccia, che indicano la direzione dell'operazione. In certi casi non è semplice tracciare queste linee; ad esempio, in diagrammi di flusso complessi, può capitare di attraversare dei blocchi, etc. In questi casi è di aiuto il simbolo di *connessione*, nel quale è posta una lettera o un nùmero: tutte le connessioni che contengono la stessa lettera o lo stesso numero sono considerate collegate fra loro. Il simbolo di *connessione di fuori pagina* svolge la stessa funzione del simbolo di connessione, solo che si usa quando il diagramma di flusso occupa più di una pagina: le connessioni di fuori pagina che contengono la stessa lettera o lo stesso numero sono considerate collegate fra loro. La presenza del simbolo di connessione di fuori pagina in luogo del simbolo di connessione segnala a chi legge che il diagramma occupa più di una pagina, e che i due (o più) punti da collegare sono raffigurati in pagine diverse.

Vediamo ora un altro esempio di diagramma di flusso. Nel paragrafo 1-3 abbiamo esaminato un programma che calcolava le radici dell'equazione di secondo grado

$$ax^2 + bx + c = 0 \qquad (4\text{-}14)$$

dove le radici sono

$$x_1 = \frac{-b + \sqrt{b^2-4ac}}{2a} \qquad (4\text{-}15a)$$

$$x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a} \qquad (4\text{-}15b)$$

Se $b^2 - 4ac$ è negativo, sorge un problema, perché, in BASIC, non si può ricavare la radice quadrata di un numero negativo. Ricordiamo che la radice quadrata di un numero negativo è un numero immaginario; cioè possiamo scrivere

$$\sqrt{-16} = \sqrt{-1}\sqrt{16} = 4\sqrt{-1} = 4i$$

La lettera i è qui usata per rappresentare $\sqrt{-1}$.
Vediamo un altro esempio:

$$5 + \sqrt{-64} = 5 + i8$$

Ora, se $b^2 - 4ac$ è negativo, si può scrivere

$$x_1 = \frac{-b + i\sqrt{4ac-b^2}}{2a} \qquad (4\text{-}17a)$$

$$x_2 = \frac{-b-i\sqrt{4ac-b^2}}{2a} \qquad (4\text{-}17b)$$

Si noti che $4ac - b^2$ sarà positivo se $b^2 - 4ac$ è negativo. Vediamo ora come si possa scrivere un programma che ricavi le radici dell'equazione (4-14) per qualsiasi valore di $b^2 - 4ac$, tenendo presente che, se $b^2 - 4ac = 0$, ci sono due radici uguali. Il programma è il seguente:

```
 10  REM PROGRAMMA PER CALCOLARE LE RADICI
 15  REM DI UNA EQUAZIONE QUADRATICA
 20  PRINT "INTRODURRE A,B,C"
 30  INPUT A,B,C
 40  LET D = B↑2–4*A*C
 50  IF D < = 0 THEN 100
 60  X1 = (–B+D↑.5)/(2*A)
 70  X2 = (–B–D↑.5)/(2*A)
 80  PRINT "RADICE 1 ="; X1; "RADICE 2 =";X2
 90  STOP
100  IF D< 0 THEN 140
110  X1 = –B/(2*A)
120  PRINT "RADICI UGUALI =";X1
130  STOP
140  LET D = –D
150  LET Y1 = –B/(2*A)
160  LET Y2 = D↑.5/(2*A)
170  PRINT "LE RADICI SONO";Y1;"+ O – I";Y2
180  END                                            (4-18)
```

In Figura 4-3 è rappresentato il diagramma di flusso di questo programma. Spieghiamo le fasi operative del programma ed il diagramma di flusso.

Dopo che i valori di A, B e C sono introdotti, si calcola

D = B↑2–4*A*C (equivalente a $d = b^2 - 4ac$)

Volendo ricavare la radice quadrata di D, dobbiamo sapere se D è positivo o negativo. Lo statement di IF-THEN numero 50 determina se D è positivo e corrisponde, nel diagramma di flusso, al primo riquadro di decisione. Se D è positivo, sono eseguiti gli statements 60-80, con cui sono calcolati X1 = (–B+D↑.5)/(2*A) e X2 =
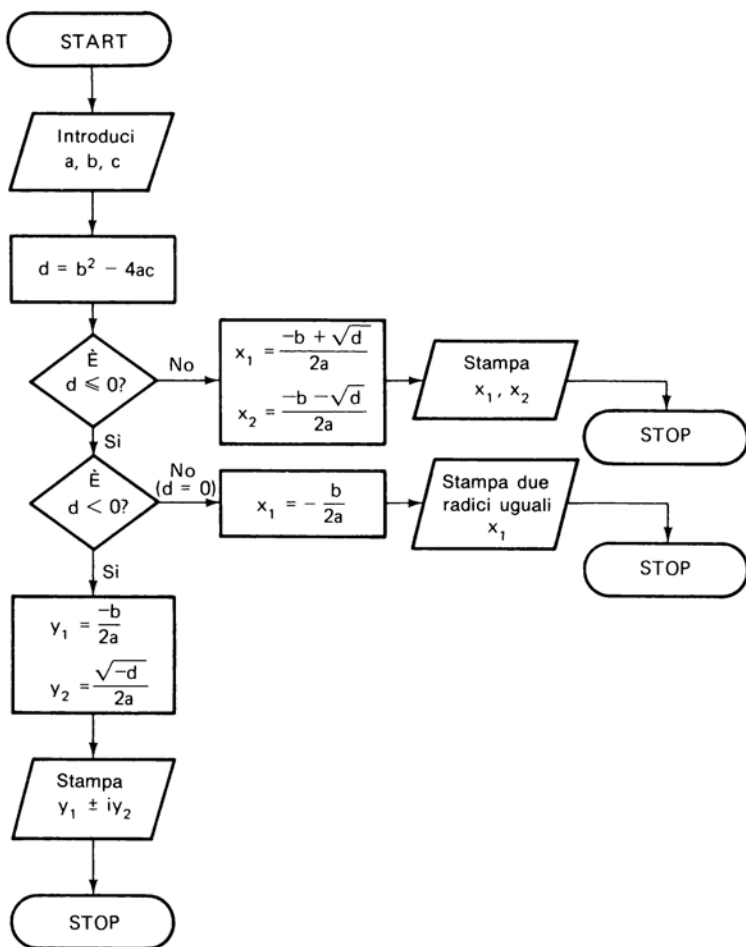
Figura 4.3 — Diagramma di flusso del programma (4-18) che calcola le radici di un'equazione di secondo grado.

(—B—D↑.5)/(2∗A), quindi sono stampati i risultati. A questo punto l'elaborazione termina. Se invece D è uguale o minore di 0, all'esecuzione dello statement 50 il controllo passerà allo statement 100, che corrisponde al secondo riquadro di decisione del diagramma di flusso. Lo statement di IF-THEN numero 100 determina se D è negativo o uguale a zero. Se D = 0, non è minore di zero, e quindi viene eseguito lo statement 110. Essendo D = 0, i valori delle due radici sono uguali, e sono dati da X1 = —B / (2∗A). A questo punto lo statement 120 fa stampare RADICI UGUALI = ed il valore di X1. Quindi l'elaborazione termina. Se invece D è negativo, all'esecuzione dello statement 100 il controllo è trasferito allo statement 140. A questo punto a D si

sostituisce −D, e si calcolano Y1 = −B/(2*A) e D↑.5/(2*A). Allora le radici vengono date da Y1 + iY2 e da Y1 − iY2: questi passaggi rappresentano i calcoli richiesti dalle equazioni (4-17). Lo statement 180 stampa le radici nella forma: Y1 ± iY2. A questo punto l'elaborazione termina.

4-5. LA DOCUMENTAZIONE

Nel momento in cui scrive un programma, il programmatore sa quale algoritmo ha usato e perché, quali dati debbano essere introdotti e qual è il risultato. Tuttavia in poco tempo questi particolari si dimenticano. Quindi è opportuno prendere delle note sul programma, tanto più che spesso il programma è scritto da un programmatore, ma utilizzato da altri che, nell'esecuzione del programma, saranno aiutati da queste note. L'insieme delle note prese, che chiamiamo *documentazione,* è estremamente utile. La documentazione dovrebbe contenere le seguenti informazioni:

1) Il nome del programma.
2) Il suo scopo, ossia il problema da esso risolto.
3) Il nome e l'indirizzo del programmatore: questo permette all'utilizzatore di mettersi in contatto con lui se sorgono dei problemi.
4) I dati d'ingresso richiesti ed il modo in cui essi devono essere forniti, ad esempio se si usano statements di INPUT. In questo caso si può dare un'informazione tipo "introdurre tre voti separati da virgole in risposta al carattere di sollecitazione".
5) La forma dei dati in uscita.
6) I dispositivi ausiliari che occorrono (telescrivente, lettore di schede, etc.).

Una volta date queste informazioni, chiunque, anche se non conosce il BASIC, è in grado di eseguire il programma.

La documentazione servirà anche ad altri programmatori che vogliano studiare il programma: pertanto dovrebbero essere descritti il programma e l'algoritmo. A tal fine nella documentazione dovranno comparire in più le seguenti informazioni:

7) Il diagramma di flusso.
8) Il programma.
9) Una descrizione verbale, come quella fatta nel paragrafo 4-4, che illustri l'algoritmo ed il programma.
10) I dati di test: questo permette all'utilizzatore di verificare se il programma è eseguito correttamente sul suo calcolatore.

In generale, una buona documentazione dà al lettore una descrizione completa di come eseguire il programma, del suo algoritmo, e del programma stesso; tutto ciò aiuterà moltissimo i programmatori, e tutti coloro che eseguiranno il programma.

4-6. LO STATEMENT DI ON-GO TO

Gli statements di GO TO e di IF-THEN trasferiscono il controllo ad un solo e ben preciso statement. Vedremo ora uno statement che ci permette di scegliere, fra diversi statements, quello a cui dev'essere passato il controllo: questo statement è detto statement di ON-GO TO. Si tenga presente che gli statements di GO TO e di IF-THEN si usano più spesso dello statement di ON-GO TO; tuttavia ci sono casi in cui quest'ultimo è estremamente utile.

Vediamo un esempio di statement di ON-GO TO:

```
50   ON A GO TO 90,60,150,140
```
(4-19)

La sua forma è: numero di linea, la parola-chiave ON, una variabile (o un'espressione), le parole GO TO, seguite da un elenco di numeri separati da virgole. Si osservi che non c'è la virgola né dopo TO né alla fine dell'elenco. I numeri dell'elenco *devono* corrispondere a numeri di statements presenti nel programma in BASIC.

L'operatività dello statement (4-13) è la seguente: se A = 1, il controllo passa allo statement 90. Se A = 2, il controllo passa allo statement 60. Allo stesso modo, se A = 3, il controllo passa allo statement 150 e se A = 4, il controllo passa allo statement 140. Come regola generale, se A = 3, il controllo passa allo statement il cui numero è il terzo dell'elenco, e così via.

Se A non è un numero intero, viene troncato (v. par. 3-7); così 3.678 sarà troncato a 3, etc.

Noi abbiamo presentato una lista con quattro numeri di statement: in realtà possono essercene di più o di meno. Supponiamo che nella lista compaiano n termini: il valore troncato di A dev'essere compreso fra 1 ed n; in caso contrario, sarà generato un messaggio di errore.

Nell'esempio precedente abbiamo illustrato lo statement di ON-GO TO servendoci di una variabile: in realtà può essere utilizzata anche un'espressione. Ad esempio,

```
50   ON A*B*B↑C+2 GO TO 40,60,100,150,180
```
(4-20)

Qui è il valore (troncato) di A+B↑C+2 a determinare a quale statement è trasferito il controllo: così, se A*B↑C+2 = 3.45, il controllo passerà allo statement 100.

4-6-1. Lo statement di ON-THEN

Molte versioni di BASIC ammettono anche uno statement che equivale allo statement di ON-GO TO. In esso, al posto delle parole GO TO, troviamo THEN. Ad esempio, il seguente statement è equivalente allo statement (4-20):

```
50   ON A*B↑C+2 THEN 40,60,100,150,180                    (4-21)
```

Si tenga presente che non tutti i compilatori BASIC ammettono questa forma, mentre tutti ammettono la forma ON - GO TO.

Come esempio consideriamo un programma in cui si usa lo statement di ON-GO TO. Supponiamo che un venditore di pezzi di ricambio di automobili venda tre diversi pezzi, numerati rispettivamente con 1, 2 e 3, e che la sua commissione su ogni pezzo sia come nella tabella seguente:

Numero del pezzo	Commissione
1	L. 2,100
2	L. 12,500
3	L. 3,980

Ad ogni vendita effettuata, il numero del pezzo e le quantità vendute sono riportati in una bolla di vendita; a fine mese i dati provenienti da tutte le bolle di vendita verranno introdotti nel calcolatore. Generalmente, il numero globale delle bolle di vendita non è conosciuto, perché non si può sapere quante vendite si faranno: desideriamo quindi un programma che calcoli la commissione globale del venditore. Supponiamo che si possa introdurre un numero arbitrario di bolle di vendita, e che l'ordine in cui esse vengono introdotte non abbia importanza. Il programma è il seguente:

```
 10  REM PROGRAMMA PER CALCOLARE
 15  REM LA COMMISSIONE
 20  PRINT "INTRODURRE IL NUMERO DEL PEZZO",
 25  PRINT "IL NUMERO GLOBALE DELLE VENDITE";
 30  LET C = 0
 40  INPUT N,S
 50  IF N = 0 THEN 900
 60  ON N GO TO 70,90,110
 70  C = C + 2,100*S
 80  GO TO 40
 90  C = C + 12,500*S
100  GO TO 40
110  C = C + 3,980*S
120  GO TO 40
900  PRINT "LA COMMISSIONE GLOBALE E'",C
999  END                                                  (4-22)
```

In Figura 4-4 è riportato il diagramma di flusso del programma.
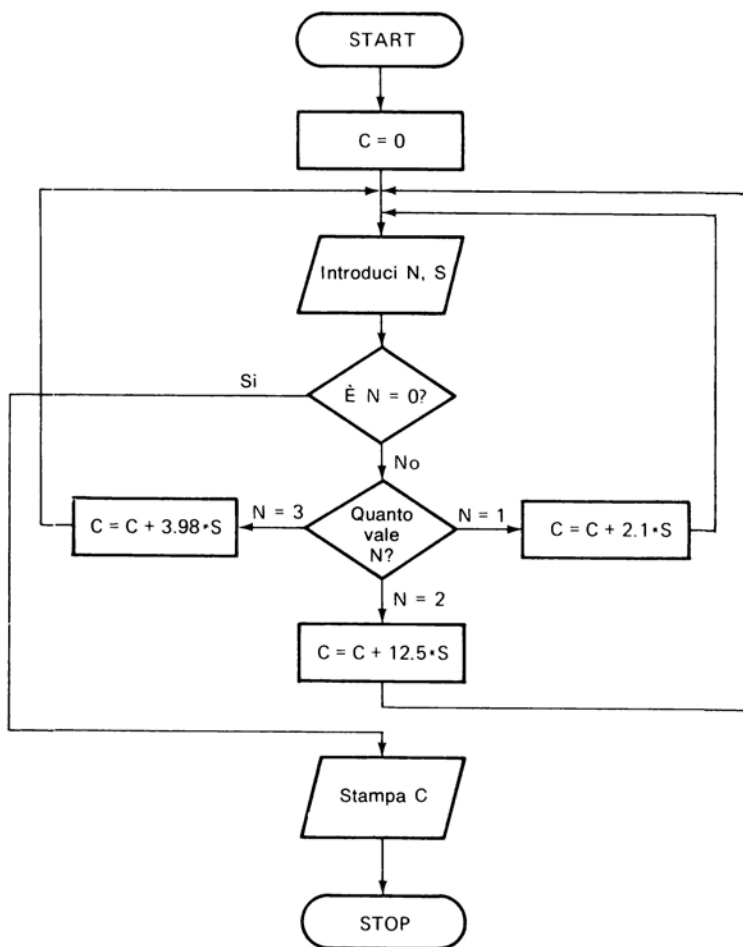
Figura 4.4 — Diagramma di flusso del programma (4-22) che calcola la commissione di un venditore. Si tenga presente che le linee alla cui intersezione non compare una freccia sono considerate non in contatto fra loro.

Esaminiamo le fasi operative del programma e/o il diagramma di flusso. N rappresenta il numero del pezzo; S indica la quantità di pezzi venduti; C la commissione globale. All'inizio, poniamo C = 0 nello statement 30, e nel primo riquadro di processo del diagramma di flusso. A questo punto vengono introdotti N ed S: se N è diverso

da zero, allora lo statement 50 sarà ignorato. Vediamo ora lo statement 60. Se N = 1, il controllo è trasferito allo statement 70. In esso abbiamo:

C = C + 2,100*S (4-23a)

Cioè il valore di C è aumentato di 2,100 volte il numero di articoli venduti del gruppo 1, in quanto la commissione per ciascun pezzo del gruppo 1 è di L. 2,100. Se N = 2, il controllo passa allo statement 90, nel quale

C = C + 12,500*S (4-23b)

Ora il valore di C è aumentato di 12,500*S, in quanto L. 12,500 è la commissione per ciascun pezzo venduto del gruppo 2. Allo stesso modo, se N = 3, il controllo passa allo statement 100, nel quale

C = C + 3,980*S

Essendo di L. 3,980 la commissione per i pezzi del gruppo 3, lo statement di GO TO ci permette di scegliere la diramazione del programma corrispondente al numero di pezzo introdotto. Il processo in cui si prende questa decisione è indicato nel diagramma di flusso dal simbolo di decisione contrassegnato con "qual è N". Una volta che lo statement 70, o 90, o 110, è eseguito, il controllo torna allo statement 40. A questo punto vengono introdotti i nuovi valori di N ed S, e la procedura si ripete; quindi C è aumentato della quantità opportuna.

Quando si sono introdotte tutte le informazioni sulle vendite, l'operatore introduce 0 per N, ed un valore qualunque per S. (Quest'ultimo valore non sarà mai usato, per cui può essere arbitrario: è necessario introdurre un valore purchessia, perché lo statement 40 richiede che vengano introdotti due dati.) Ora lo statement 50 non è ignorato, ma il controllo passa allo statement 900: a questo punto viene stampata la commissione globale e l'elaborazione ha termine.

Questo programma illustra non solo lo statement di ON-GO TO, ma anche altre due tecniche, interessanti ed utili. Primo, mostra come si possa aggiornare continuamente una variabile con nuove informazioni e come si possa ottenere una somma totale (v. le (4-23)). In altre parole, se scriviamo C = C + (nuova quantità), C assume un nuovo valore che è uguale al suo vecchio valore più la nuova quantità. Secondo, mostra come si possa far terminare l'attività di un programma al momento opportuno, mediante dei dati introdotti da telescrivente.

ESERCIZI

4-1. Illustrate l'effetto dello statement seguente:

    90 GO TO 30

4-2. Determinate quale valore di A verrà stampato in seguito all'esecuzione del seguente segmento di programma scritto in BASIC:

```
     .
20   LET A = 2.3
30   LET B = 5
40   GO TO 60
50   A = A*B
60   PRINT A
```

Verificate il risultato ottenuto scrivendo un programma opporturno ed eseguendolo su di un calcolatore.

4-3. Modificate il programma dell'Esercizio 2-26 in modo che il programma si ripeta automaticamente.

4-4. Modificate il programma dell'Esercizio 4-3 in modo che l'elaborazione termini automaticamente quando il programma esaurisce i dati.

4-5. Illustrate l'effetto dello statement seguente:

```
40 IF A = B+2 THEN 80
```

4-6. Ripetete l'Esercizio 4-5 con

```
40 IF A< > B*A THEN 120
```

4-7. Illustrate il significato di tutti gli operatori di relazione della Tabella 4-1.

4-8. Considerate il segmento di programma

```
     .
40   IF A > B↑2 THEN 70
50   A = A + 2*B
60   GO TO 80
70   A = 2*A
80   PRINT A
```

Se i valori, calcolati in precedenza, di A e B sono A = 26 e B = 4, quale valore di A sarà stampato? Verificate il risultato ottenuto scrivendo un programma in BASIC ed eseguendolo su di un calcolatore.

4-9. Ripetete l'Esercizio 4-8, ma questa volta con A = 2, e B = 4.

4-10. La commissione di un venditore si basa sul totale delle sue vendite nel modo seguente: per ogni pezzo venduto di prezzo inferiore a L. 100,000, la commissione è del 10%; per ogni pezzo venduto di prezzo compreso fra L. 100,000 e L. 300,000, la commissione è dell'8%; per ogni pezzo venduto di prezzo superiore a L. 300,000, la commissione è del 12%. Scrivete un programma che calcoli la commissione quando è introdotto il totale delle vendite.

4-11. Scrivete un programma che calcoli la funzione

$$f = x^3 + 4x^2 + g$$

dove

$$g = 1 \text{ se } x^3 + 2x^2 > 15$$

e

$$g = 10 \text{ se } x^3 + 2x^2 \leqslant 15$$

Eseguite il programma con i valori $x = 1$ ed $x = 10$.

4-12. Ripetete l'Esercizio 4-11, ma dando questa volta ad x i valori 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Fate in modo che questo avvenga automaticamente, cioè non rieseguite il programma per dieci volte.

4-13. Disegnate il diagramma di flusso del programma dell'Esercizio 4-10.

4-14. Disegnate il diagramma di flusso del programma dell'Esercizio 4-11.

4-15. Disegnate il diagramma di flusso del programma dell'Esercizio 4-12.

4-16. Illustrate gli effetti dello statement seguente:

```
50   ON A GO TO 10,20,30,60
```

4-17. Ripetete l'Esercizio 4-16 con

```
70   ON A*B↑2 GO TO 80,100,250,300
```

4-18. Determinate quale valore di A risulterà eseguendo il seguente segmento di programma in BASIC:

```
50   ON A*B GO TO 60,70,80
60   A = A + 2*B
70   GO TO 90
80   A = 2.36
90   PRINT A
```

I valori assegnati in precedenza siano: A = 0.75, e B = 1.5. Verificate i risultati ottenuti scrivendo un opportuno programma in BASIC ed eseguendolo su di un calcolatore.

4-19. Ripetete l'Esercizio 4-8, ma ponendo A = 1 e B = 2.

4-20. Ripetete l'Esercizio 4-18, ma ponendo A = 1.6 e B = 2.

4-21. Scrivete un programma che esegua le seguenti operazioni: introdotti in un ordine qualsiasi i voti riportati da uno studente in sei prove, si deve calcolare la media dei voti. Se la media è minore di 60, dovrà essere stampato un messaggio di ammonimento; se la media è maggiore di 90, dovrà essere stampato un messaggio di elogio per lo studente. Dovranno inoltre essere stampati la media, il voto più alto e quello più basso. Disegnate poi il diagramma di flusso del programma.

4-22. Gli studenti di una classe fanno cinque esercitazioni. La loro media è una media pesata, data da

$$\text{MEDIA} = \frac{2\ (\text{prova } 1) + (\text{prova } 2) + 3\ (\text{prova } 3) + 1.5\ (\text{prova } 4) + 5\ (\text{prova } 5)}{8}$$

I voti delle esercitazioni possono essere introdotti in un ordine qualunque; ciascuna coppia di dati (numero dell'esercitazione-voto) dev'essere introdotta insieme. Scrivete un programma che calcoli la media pesata.

CAPITOLO 5

PROCEDURE CICLICHE

In questo capitolo analizzeremo degli statements che si rivelano estremamente vantaggiosi quando si vuole che il programma ritorni ciclicamente su sé stesso. Realizzare una procedura ciclica è utilissimo, in quanto porta ad una semplificazione dei programmi. Nel prossimo capitolo tratteremo di una procedura di programmazione che, usata insieme con la procedura ciclica, estende notevolmente la versatilità della programmazione.

5-1. CONCETTI FONDAMENTALI SUI CICLI

Nella programmazione spesso ci occorre che il programma ritorni ciclicamente su sé stesso: questo accade sovente quando lo stesso tipo di calcolo viene ripetuto più volte. Ormai conosciamo un numero sufficiente di statements del BASIC per inserire un *ciclo* (*loop* in inglese) in un programma. Vediamo come si procede.

Supponiamo di voler calcolare il fattoriale di un numero intero, che, come sappiamo, è dato da

$$n! = (1)(2)\ \dots\ (n-1)(n) \qquad (5\text{-}1)$$

Ad esempio,

$$6! = 1x2x3x4x5x6 = 720$$

Vogliamo scrivere un programma nel quale il valore di n è introdotto da tastiera, e che in seguito calcola n!. Il programma sarà il seguente:

```
10  REM PROGRAMMA PER CALCOLARE
15  REM IL FATTORIALE
```

```
 20  PRINT "INTRODURRE N";
 30  INPUT N
 40  LET K = 1
 50  LET F = 1
 60  LET F = F*K
 70  LET K = K+1
 80  IF K < = N THEN 60
 90  PRINT F
100  END
```

(5-2)
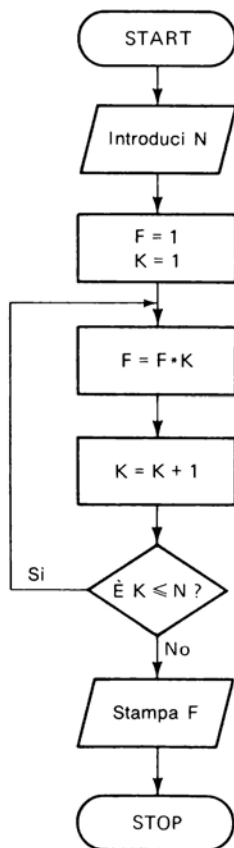
Figura 5.1 — Diagramma di flusso del programma (5-1) che calcola N!.

Esaminiamone le fasi operative, aiutandoci con la Figura 5-1, in cui è rappresentato il relativo diagramma di flusso. Gli statements 20-50 provvedono all'introduzio-

ne del valore di N, ed a porre il valore iniziale delle due variabili F e K uguale ad 1: alla fine F sarà il valore del fattoriale. Nello statement 60 poniamo F = F*K, mentre nello statement 70 incrementiamo di 1 il valore di K. Se il valore incrementato di K è minore o uguale ad N, il controllo torna allo statement 60: gli statements 60-80 costituiscono un ciclo. Esaminiamone l'andamento giro per giro. Nel primo giro K ed F e, di conseguenza, F*K, sono tutti uguali ad 1; poi K viene incrementato di 1. Pertanto, quando lo statement 60 è eseguito per la seconda volta, F*K = (1)(2). Poi K è ancora una volta incrementato di 1 e, al successivo giro del ciclo, all'esecuzione dello statement 60 F diventa (1)(2)(3). K viene quindi di nuovo incrementato di 1, ed al giro successivo F diventa (1)(2)(3)(4). Questo andamento ciclico continua finché K diventa maggiore di N. Dal momento che K ed N sono numeri interi, il programma calcola (1)(2)...N come per l'appunto si voleva.

Abbiamo mostrato come si possa risolvere un certo problema ricorrendo ad una procedura ciclica. Questa tecnica si usa molto spesso. Il BASIC ha comunque alcuni statements speciali che realizzano una procedura ciclica molto facilmente e vantaggiosamente: si tratta dello statement di FOR-TO e dello statement di NEXT. Illustriamoli con un esempio:

```
      .
 40   FOR I = 1 TO 10
 50   LET F = I*J + F

      .
 80   ----------
 90   NEXT I
100   ----------                                   (5-3)
```

Esaminiamone le fasi operative, allo scopo di vedere la forma esatta di questi statements. All'esecuzione dello statement 40, la variabile I è posta uguale ad 1. Il controllo quindi passa attraverso tutti gli statements successivi allo statement 40 con 1 come valore di I. Giunti allo statement 90, I viene incrementato di 1 (cioè I diventa 2), ed il controllo ritorna allo statement che segue lo statement di FOR-TO. A questo punto il controllo prosegue attraverso gli statements 50-80, con I = 2. Arrivati di nuovo allo statement di NEXT, I viene ancora incrementato di 1, ed il controllo ritorna allo statement che vien dopo lo statement di FOR-TO. Il ciclo continua in questo modo finché il valore di I incrementato diventa maggiore di 10 (l'ultimo numero che compare nello statement di FOR-TO): allora il ciclo cessa, e viene eseguito lo statement che segue lo statement di NEXT I, cioè lo statement 100.

Lo statement di FOR-TO ha una forma più generale di quella che si è vista qui: in-

fatti vi si possono utilizzare delle variabili. Ad esempio, possiamo scrivere gli statements di FOR-TO e di NEXT in questo modo:

```
        .
40   FOR M = N TO K
        .
90   NEXT M                                        (5-4)
```

dove ad M e a K deve essere assegnato un valore prima dello statement di FOR-TO. La forma dello statement di FOR-TO è: numero di linea, la parola-chiave FOR, una variabile, detta *variabile di esecuzione*, il segno uguale, un numero o una variabile, detti *valore iniziale*, la parola TO, seguita da un altro numero o variabile, detti *valore finale*. (Più avanti, in questo stesso paragrafo, parleremo più diffusamente di questo punto.) Lo statement di NEXT ha la forma: numero di linea, la parola-chiave NEXT, seguita da una variabile che *deve* essere la stessa della variabile di esecuzione dello statement di FOR-TO. Si noti che sia nello statement di FOR-TO che nello statement di NEXT non compaiono virgole. Un ciclo è formato dagli statements che seguono lo statement di FOR-TO fino allo statement di NEXT.

Consideriamo l'esempio (5-4): il ciclo parte con M = N, e, ogni volta che compie un giro, M viene incrementato di 1. L'ultimo giro del ciclo è quello nel quale M = K.

Vediamo il funzionamento di questa procedura ciclica inserendola nel programma (5-2):

```
10   REM PROGRAMMA PER CALCOLARE
15   REM IL FATTORIALE
20   PRINT "INTRODURRE N";
30   INPUT N
40   LET F = 1
50   FOR K = 1 TO N
60   F = F*K
70   NEXT K
80   PRINT F
90   END                                           (5-5)
```

Esaminiamone le fasi operative ed il diagramma di flusso, rappresentato in Figura 5-2a. I concetti di base sono in gran parte gli stessi del programma (5-2), e pertanto non li ripeteremo qui. Negli statements 30 e 40 viene introdotto il valore di N, ed il valore iniziale di F è posto uguale ad 1. Si noti che non dobbiamo inizializzare K (in

questo caso si deve porre K = 1), perché lo si farà nello statement di FORT-TO. Il ciclo è dato dalle linee 50-70. K = 1 al primo giro, ed è incrementato di 1 ad ogni ripetizione del ciclo: pertanto, quando K = 3, F, dopo l'esecuzione dello statement 60, sarà uguale a (1) (2) (3). L'ultimo giro è quello in cui K = N. A questo punto N! sarà calcolato come nel programma (5-2).
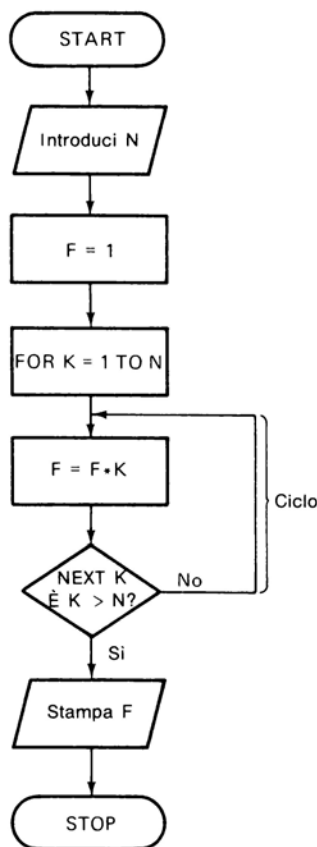
Figura 5.2a — Diagramma di flusso del programma (5-5) che calcola N!.

La procedura di esecuzione di un ciclo può variare in maniera minima da una versione di BASIC ad un'altra: possiamo vedere queste variazioni confrontando le Figure 5-2a e 5-2b. Entrambe illustrano il programma fattoriale (5-5), e sono sostanzialmente uguali, tranne che per la collocazione del blocco di decisione con il quale è verificato il momento in cui il ciclo termina (cioè quando K > N). In Figura 5-2a il blocco di decisione si trova alla fine del ciclo, mentre in Figura 5-2b è all'inizio. Questo fatto non incide sul funzionamento del programma, tranne che in un caso parti-

colare. Supponiamo d'introdurre un valore per N che sia minore di 1 (valore iniziale): la Figura 5-2a ci dice che lo statement 60 (LET F = F*K) verrà calcolato una sola volta, e quindi il ciclo terminerà, mentre la Figura 5-2b ci dice che lo statement 60 non verrà mai calcolato. Nel programma (5-5), in entrambi i casi il risultato sarà lo stesso, ma in altri programmi si potranno ottenere risultati differenti.
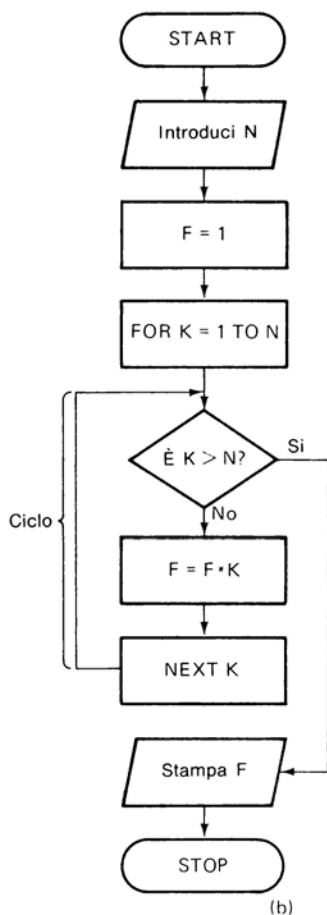
Figura 5.2b — Un altro diagramma di flusso del programma (5-5).

Vediamo queste due diverse realizzazioni degli statements di FOR-TO e di NEXT. La realizzazione di Figura 5-2a indica che, se il valore iniziale è maggiore del valore finale, il ciclo compirà un solo giro; la Figura 5-2b invece mostra che, in questo stesso caso, non ci sarà ciclo. Potrete vedere nel manuale di BASIC del calcolatore usa-

to qual è la procedura che si utilizza. I diagrammi di flusso presenti in questo libro seguono tutti la forma della Figura 5-2a.

Gli statements di FOR-TO e di NEXT costituiscono un metodo semplice per effettuare una procedura ciclica. In realtà, la procedura è più flessibile di quanto si è detto finora. La forma generale di questi statements è:

```
20  FOR K = N1 TO N2 STEP N3
30  ----------
      .
90  NEXT K                                    (5-6)
```

La forma dello statement di FOR-TO generalizzato è: numero di linea, la parola-chiave FOR, variabile di esecuzione, segno uguale, numero o variabile (valore iniziale), la parola-chiave TO, numero o variabile (valore finale), la parola-chiave STEP, seguita da un numero o una variabile, detti *lunghezza del passo.* Lo statement di NEXT è formato da: numero di linea, la parola-chiave NEXT, seguita da una variabile, che *deve* essere la stessa della variabile di esecuzione.

Vediamo quello che fanno gli statements (5-6). Il ciclo avrà luogo dallo statement che vien dopo lo statement di FORT-TO (lo statement 30 nel nostro esempio) fino allo statement di NEXT (lo statement 90). Al primo giro del ciclo, viene usato il valore K = N1; al secondo giro, K è incrementato di N3 (K = K + N3): in sintesi, ad ogni giro del ciclo, il valore di K aumenta di N3. La procedura ciclica prosegue finché K diventa maggiore di N2: a questo punto termina. Si noti che il valore di K maggiore di N2 non interviene nel ciclo. Quando il ciclo ha termine, il controllo passa allo statement che vien dopo lo statement di NEXT. Si tenga presente che, se non scriviamo STEP N3 nello statement di FORT-TO, è come se scrivessimo STEP 1. In altre parole, se non si dà lo STEP, è sottinteso che la lunghezza del passo è 1.

Non è indispensabile che i valori di N1, N2 ed N3 siano numeri interi. Consideriamo ad esempio

```
20  FOR A = 0.5 TO 10 STEP 2
      .
50  NEXT A
```

I valori di A che saranno usati nei giri successivi sono: 0.5, 2.5, 4.5, 6.5 e 8.5, mentre il successivo valore di A (cioè 10.5), essendo maggiore di 10, non verrà usato. Cioè, il ciclo termina *dopo* il giro in cui A = 8.5. Come regola generale, se la lun-

ghezza del passo è positiva, nel caso del segmento di programma (5-6) il ciclo sarà effettuato con tutti i valori di K che soddisfano la relazione

$$N1 \leqslant K \leqslant N2 \qquad (5\text{-}7)$$

cioè con i valori compresi fra N1 ed N2, inclusi N1 ed N2. Si noti che N2, che è detto valore finale, in realtà può anche non essere l'ultimo valore assunto dalla variabile di esecuzione: nell'esempio precedente, in cui N2 valeva 10, l'ultimo valore assunto dalla variabile di esecuzione è stato 8.5

Lo STEP può essere negativo. Ad esempio, il seguente statement è corretto:

```
20   FOR B = 0.5 TO -10 STEP -2                    (5-8)
```

Qui i valori di B che verranno usati nei giri successivi del ciclo sono 0.5, −1.5, −3.5, −5.5, −7.5, e −9.5. Come nel caso visto prima, −11.5 non viene usato, perché non rientra nell'intervallo di validità del ciclo. Come regola generale, e con riferimento alla notazione del programma (5-6), se il passo è negativo, il ciclo sarà eseguito con valori di K compresi nell'intervallo

$$N2 \leqslant K \leqslant N1 \qquad (5\text{-}9)$$

(Si tenga presente che −9 è più grande di −10, e così via.) Per illustrare questi concetti, esaminiamo il seguente programma:

```
10   REM PROGRAMMA CAMPIONE
20   READ A,B,C
30   LET I = 0
40   FOR D = A TO B STEP C
50   I = I + D
60   NEXT D
70   PRINT I
80   DATA .707,1.867,.5
99   END                                           (5-10)
```

Il valore di I è posto inizialmente uguale a 0 nello statement 30. I valori di A, B e C vengono forniti come dati. Ad ogni giro del ciclo, D assumerà i valori seguenti: 0.707, 1.207 e 1.707. Il valore successivo, 2.207, non rientra nell'intervallo previsto, e quindi il valore finale di D è 0.707 + 1.207 + 1.707 = 3.621. È questo il valore che sarà stampato.

La variabile che segue la parola-chiave FOR è detta variabile di esecuzione del ciclo. All'interno di un ciclo il valore di questa variabile può essere usato, ma a destra

del segno di uguale, essendo vietato adoperare statements che ne modifichino il valore: in altre parole, la variabile di esecuzione non deve comparire, all'interno del ciclo, a sinistra dell'uguale. Ad esempio, nel programma (5-10), il seguente statement:

```
55   LET D = B + .3                                    (5-11)
```

darebbe luogo ad un ciclo errato, perché il valore della variabile di esecuzione verrebbe modificato all'interno del ciclo stesso. Ci sono poi altre condizioni particolari di cui occorre tener conto. Esaminiamo uno statement di FOR-TO della forma

```
60   FOR K = N1 TO N2 STEP N3                          (5-12)
```

Se N1 ed N2 sono uguali ed N3 è diverso da zero, il ciclo compirà un solo giro, con il valore K = N1.

In molte versioni di BASIC il ciclo *non sarà affatto preso in considerazione* nei seguenti casi: 1) quando i valori iniziale e finale del ciclo sono uguali e la lunghezza del passo N3 è zero; 2) quando il valore finale N2 è maggiore del valore iniziale N1, e la lunghezza del passo è negativa; 3) quando il valore finale N2 è minore del valore iniziale N1, e la lunghezza del passo N3 è positiva. Se la lunghezza del passo è posta uguale a zero e i valori iniziale e finale non sono uguali, il ciclo continuerà ad essere eseguito all'infinito. Pertanto bisogna stare attenti ad evitare queste condizioni.

Il valore iniziale N1, il valore finale N2 e la lunghezza del passo N3 possono anche essere delle espressioni. Ad esempio, questo è uno statement BASIC corretto:

```
70   FOR K = A*D TO A–2 STEP C+D                       (5-13)
```

Ad ogni giro del ciclo, saranno calcolati A–2 e C+D. Se i valori di A, C e D non vengono modificati nel ciclo, allora si avrà uno spreco di elaborazione. Ad esempio, se A, C e D non vengono modificati durante la procedura ciclica, sarà più conveniente procedere così nella scrittura del programma:

```
65   LET N2 = A–2
67   LET N3 = C+D
70   FOR K = A*B TO N2 STEP N3                         (5-14)
```

Si noti che non è necessario calcolare N1 prima dello statement 70, perché N1 viene calcolato un'unica volta, allo scopo di ottenere il valore iniziale.

Alcune versioni di BASIC ammettono una variazione della forma dello statement di FORT-TO, per la quale la parola STEP può essere sostituita da BY. Ad esempio, i due statements che seguono:

```
70   FOR K = N1 TO N2 STEP N3                                    (5-15a)
```

e

```
70   FOR K = N1 TO N2 BY N3                                      (5-15b)
```

saranno equivalenti.

Si tenga presente che, mentre tutte le versioni di BASIC ammettono la forma (5-15a), solo alcune ammettono la forma (5-15b).

5-2. STATEMENTS DI SALTO ABBINATI AD UN CICLO

Gli statements di salto, come IF-THEN e GO TO, si possono usare abbinati ad un ciclo. Il controllo può essere trasferito sia a statements compresi nel ciclo, sia a statements al di fuori del ciclo stesso: in quest'ultimo caso, il ciclo ha termine. Si ricordi che, a meno di trovare uno statement di salto, il controllo procede da uno statement a quello che segue nell'ordine sequenziale.

Per illustrare un semplice caso di salto all'interno di un ciclo, consideriamo il seguente segmento di programma in BASIC:

```
      .
 80   REM SALTO ENTRO UN CICLO
 90   FOR I = 1 TO 100 STEP 2
100   LET A = I
110   LET A = A+5
120   IF 2*A + B>10 THEN 140
130   LET A = A+3
140   NEXT I
```

(5-16)

Si suppone che il valore di B sia già stato calcolato in un punto precedente del programma. In Figura 5-3 si può vedere il diagramma di flusso di questo program-
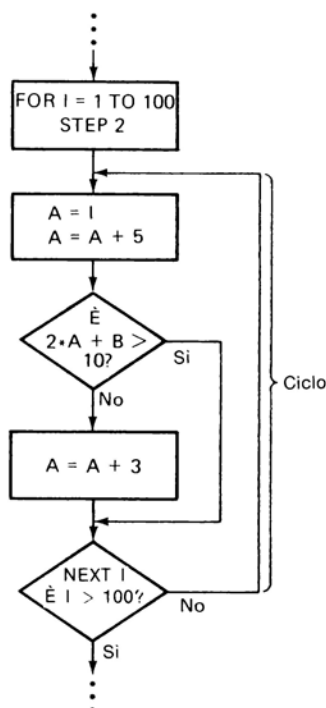
Figura 5.3 — Diagramma di flusso del segmento di programma (5-16). Si assume che B venga calcolato prima dell'esecuzione di questo segmento.

ma. Il ciclo viene impostato mediante gli statements 90 e 140. Vediamo come opera. Nello statement 90 definiamo una variabile A, il cui valore è uguale ad I. Questo può sembrare un passaggio non necessario: perché non usare addirittura I in luogo di A? Non lo si fa perché I è la variabile di esecuzione del ciclo, ed il suo valore *non deve* essere modificato all'interno del ciclo stesso. Si noti che nello statement 110 cambiamo il valore di A (A = A+5): questo è lecito, mentre sarebbe *sbagliato* scrivere I = I+5. Di fatto abbiamo scritto qui uno statement in più, perché gli statements 100 e 110 si potrebbero combinare in un unico statement, e cioè:

```
100   LET A = I+5                                              (5-17)
```

Nel segmento di programma (5-16) abbiamo inserito due statements per sottolineare il fatto che la variabile di esecuzione non dev'essere modificata all'interno del ciclo.

Vediamo ora lo statement 120. Se 2*A+B >10, il controllo sarà trasferito allo statement 140. (Supponiamo che B sia già stato calcolato in un punto precedente del

programma.) Quando lo statement 140 è eseguito, I viene incrementato della lunghezza del passo, cioè di 2, e parte il successivo giro del ciclo: si tenga presente che, per dare inizio ad un nuovo giro del ciclo, il controllo deve passare allo statement di NEXT, e non allo statement di FOR-TO.

Se 2*A+B non è maggiore di 10, lo statement 130 verrà eseguito prima dell'esecuzione di un nuovo giro del ciclo.

La sequenza di programma (5-16) illustra un salto che avviene all'interno del ciclo, cioè che non fa uscire il controllo dall'ambito del ciclo stesso. Ma possiamo avere anche un salto che trasferisce il controllo fuori del ciclo. Consideriamo il seguente segmento di programma:

```
 50  FOR R = 1 TO 50 STEP 3
 60  LET A = R
 70  LET B1 = R*2 + R↑2
 80  IF R+B>50 THEN 110
 90  LET B1 = B1↑.5
100  NEXT R
110  LET B2 = B1+4
120  LET B3 = B2+A
```

(5-18)

Esaminiamone le fasi operative, supponendo che il valore di B sia stato introdotto in precedenza. In Figura 5-4 è riportato il diagramma di flusso relativo. Ignoriamo per il momento lo statement 60. Nello statement 70 viene calcolato il valore di B1 come funzione della variabile di esecuzione del ciclo, R. Lo statement 80 è uno statement di IF-THEN. (Abbiamo supposto che B sia stato calcolato prima di questo segmento di programma.) Se R+ B è uguale o minore di 50, questo segmento viene ignorato; se invece R+B è maggiore di 50, il controllo passa allo statement 110. *Questo statement si trova al di fuori dell'intervallo del ciclo*, per cui a questo punto l'attività del ciclo stesso termina. Questo avviene dopo che è eseguito lo statement 110; poi lo statement 120 sarà eseguito, e così via. Si tenga presente che una procedura ciclica può terminare in due modi: il primo è quello ordinario, cioè, come nel caso (5-18), quando il valore della variabile di esecuzione R supera 50; il secondo modo si ha quando il controllo viene trasferito al di fuori del ciclo stesso dallo statement di IF-THEN.

Il valore della variabile di esecuzione sarà memorizzato, per cui potrà essere usato in seguito in altri calcoli richiesti dal programma. Comunque, nel caso che il ciclo termini nel modo ordinario, il valore memorizzato varia da un calcolatore all'altro.
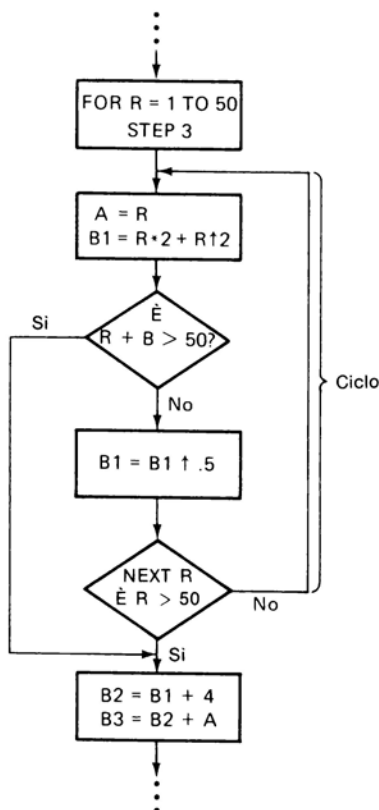
Figura 5-4 — Diagramma di flusso del segmento di programma (5-18). Si assume che B venga calcolato prima dell'esecuzione di questo segmento.

Per evitare problemi di questo tipo, si può definire una nuova variabile uguale alla variabile di esecuzione, come è fatto nello statement 60, nel quale si pone A = R: il valore di A sarà sempre uguale all'ultimo valore usato di R, per cui A può venir utilizzata in tutti i calcoli successivi.

Vediamo ora un altro esempio che illustra la procedura ciclica associata ad un trasferimento del controllo. Scriviamo un programma che esegua la somma

$$\sum_{k=1}^{100} \frac{1}{a^k} = \frac{1}{a} + \frac{1}{a^2} + \frac{1}{a^3} + \dots + \frac{1}{a^{100}} \qquad (5\text{-}19)$$

Il valore di a dev'essere introdotto dall'utilizzatore. Ad esempio, se a = 3, si ha:

$$\sum_{k=1+1}^{100} = \frac{1}{3} + \frac{1}{9} + \frac{1}{27} + \frac{1}{81} + \frac{1}{243} + \frac{1}{729} + \frac{1}{2187} + \ldots + \frac{1}{3^{100}} \qquad (5\text{-}20)$$

(Si noti che le equazioni (5-13) e (5-20) spiegano il segno Σ.)

I termini diventano molto piccoli all'aumentare di k; non tenendo conto nei calcoli dei termini molto piccoli, si avrà una perdita di precisione assolutamente trascurabile, non solo, ma si risparmierà tempo di calcolatore. Scriviamo perciò un programma che termini l'elaborazione allorquando uno qualunque dei termini diventa minore di n, ove anche n è un numero che dev'essere introdotto dall'utilizzatore. Ad esempio, per n = 0.001, la somma (5-20) diventerà:

$$\frac{1}{3} + \frac{1}{9} + \frac{1}{27} + \frac{1}{81} + \frac{1}{243} + \frac{1}{729}$$

Un programma che esegue la somma desiderata è il seguente:

```
 10  REM PROGRAMMA PER SOMMARE 1/A↑K
 15  REM CON TOLLERANZA N
 20  PRINT "INTRODURRE A E LA TOLLERANZA";
 30  INPUT A,N
 40  LET Y = 1/A
 50  LET S = Y
 60  FOR I = 1 TO 99
 70  LET Y = Y/A
 80  IF Y < N THEN 110
 90  LET S = S+Y
100  NEXT I
110  PRINT S
120  END                                                    (5-21)
```

Le fasi operative del programma, il cui diagramma di flusso è in Figura 5-5, sono le seguenti: vengono introdotti A ed N, quindi si calcola Y = 1/A e si pone S = Y. Il valore finale di S sarà la somma richiesta. A questo punto facciamo partire il ciclo, ignorando, per il momento, lo statement di IF-THEN numero 80. All'inizio Y è sostituito da Y/A. Si noti che ciascun termine della sommatoria si può ricavare da quello precedente, dividendo quest'ultimo per A (v. l'equazione (5-19)): quindi lo statement 70 fornisce il termine successivo della sommatoria. Poi, nello statement 90, S è incrementato di questo valore. A questo punto, se si sono compiuti tutti i giri del ciclo, si otterrà la sommatoria (5-19), che verrà quindi stampata. Si noti che il ciclo e-

volve solo per I uguale a valori che vanno da 1 a 99, perché il primo termine della sommatoria è calcolato prima dell'esecuzione del ciclo.

Esaminiamo ora lo statement 80. Se, durante un qualunque giro del ciclo, Y diventa minore della tolleranza N, il controllo sarà trasferito allo statement 110: il procedimento avrà termine e sarà stampato il risultato. Dopo di che l'eleborazione terminerà.
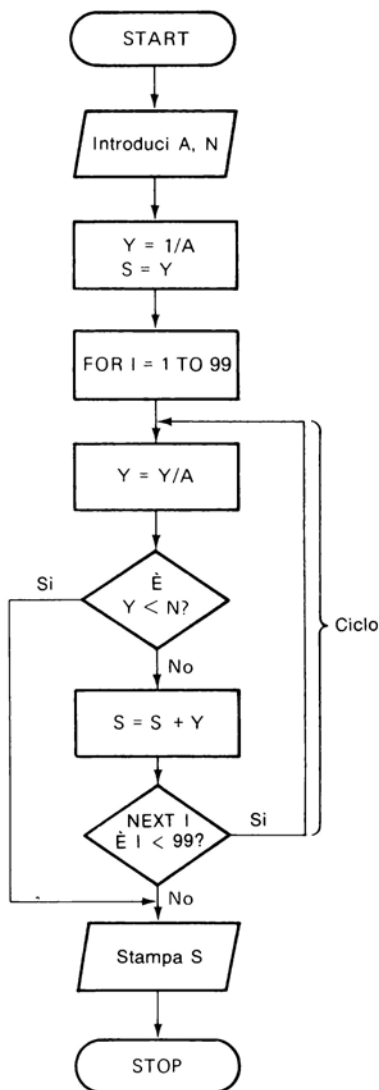
Figura 5.5 — Diagramma di flusso del programma 5-20

Si osservi che il valore di I non viene mai usato in nessun calcolo: nel caso in esame, viene utilizzato dal calcolatore semplicemente per tenere il conto del numero dei giri del ciclo che sono stati eseguiti.

Finora abbiamo illustrato il trasferimento del controllo al di fuori del ciclo: come regola generale, il controllo non può essere trasferito da uno statement che si trova al di fuori del ciclo ad uno interno al ciclo stesso. (Alcune versioni di BASIC lo ammettono, ma, quando questo avviene, la variabile di esecuzione dev'essere posta ad un valore appropriato.)

5-3. ANNIDAMENTO DI CICLI

In molti casi è utile avere un ciclo inserito in un altro ciclo: si parla allora di *cicli annidati* (*nested loops* in inglese). Esaminiamoli; in seguito formuleremo alcune regole d'uso.

```
  5  REM ESEMPIO DI CICLI ANNIDATI
 10  LET A = 0
 20  LET B = 1.2
 30  FOR K = 1 TO 15 STEP 2
 40  LET A = A+K
 50  FOR J = 3 TO 12 STEP 3
 60  LET A = A+J*B*K
 70  NEXT J
 80  NEXT K
 90  PRINT A
100  END                                    (5-22)
```

In Figura 5-6 è rappresentato il diagramma di flusso di questo programma: vediamone le fasi operative. I valori iniziali di A e di B vengono stabiliti nelle linee 10 e 20. Ci sono poi due cicli, uno interno ed uno esterno: quello esterno è definito dalle linee 30 ed 80, quello interno dalle linee 50 e 70.

Esaminiamoli. In seguito all'esecuzione dello statement 30, il valore di K è posto uguale ad 1, quindi viene eseguito lo statement 40, poi lo statement 50: questo dà il via al ciclo interno, che viene eseguito completamente mentre il valore di K resta 1. Infatti nei giri successivi del ciclo interno abbiamo K = 1, J = 3; K = 1, J = 6; K = 1, J = 9; K = 1, J = 12. A questo punto il controllo lascia il ciclo interno, perché qualsiasi ulteriore incremento di J farebbe diventare J maggiore del suo valore finale. È allora eseguito lo statement 80, che attiva il secondo giro del ciclo esterno: K assume ora il valore 3 ed il controllo torna allo statement 40, che viene eseguito. Ci si imbatte di nuovo nel ciclo interno, che compie ancora una serie di giri con K = 3; esat-
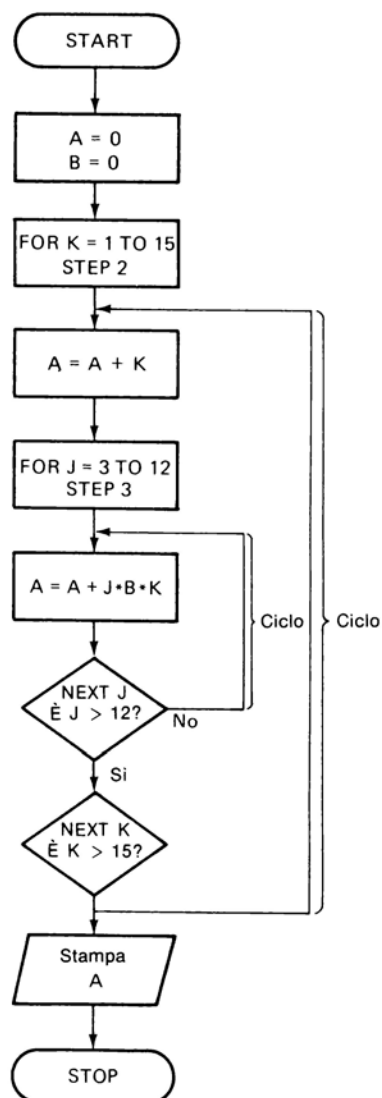
Figura 5.6 - Diagramma di flusso del programma (5-22) che illustra l'annidamento dei cicli.

tamente, compie quattro giri, con i seguenti valori: K = 3, J = 3; K = 3, J = 6; K = 3, J = 9; K = 3, J = 12. A questo punto il controllo lascia il ciclo interno, ed è eseguito lo statement 80, per cui torna in funzione il ciclo esterno con K = 5. Il procedimento fin qui descritto si ripete finché non viene usato il valore K = 15, nel senso che l'ultima attivazione del ciclo interno avverrà con i seguenti valori: K = 15, J = 3; K = 15,

J = 6; K = 15, J = 9; K = 15, J = 12. Allora il controllo lascia il ciclo interno, ma, dal momento che qualsiasi ulteriore incremento di K farebbe diventare K maggiore del suo valore finale (15), il controllo a questo punto lascia anche il ciclo esterno e passa allo statement 90. Viene stampato A e la procedura termina.

Quando i cicli sono annidati, *ogni ciclo interno dev'essere completamente contenuto nel ciclo esterno*, il che vuol dire che lo statement di NEXT relativo al ciclo interno *deve* comparire prima dello statement di NEXT relativo al ciclo esterno.

Le variabili di esecuzione *devono* essere diverse per i cicli interno ed esterno e, naturalmente, devono essere diversi anche gli statements di NEXT relativi. In Figura 5-7 sono rappresentati un annidamento di cicli corretto ed uno errato.
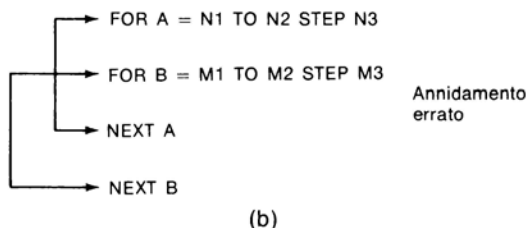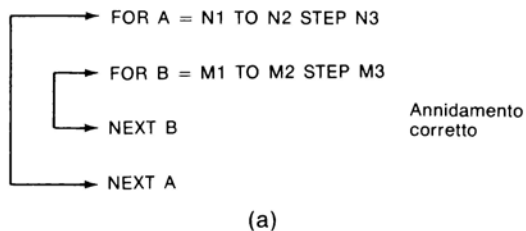
Figura 5.7 — a) Annidamento di cicli corretto;
b) Annidamento di cicli errato.

Il valore iniziale, il valore finale e la lunghezza del passo di entrambi i cicli possono essere delle variabili: in particolare, questi parametri riferiti al ciclo interno possono essere funzione della variabile di esecuzione del ciclo esterno. Consideriamo il seguente segmento di programma:

```
    .
20  FOR A = M1 TO M2 STEP M3
```

```
50   FOR B = A TO A*2 STEP A/5
        .
90   NEXT B
        .
120  NEXT A
```

(5-23)

Si osservi che il valore iniziale, il valore finale e la lunghezza del passo del ciclo interno dipendono da A, che è la variabile di esecuzione del ciclo esterno: allora il valore iniziale, il valore finale e la lunghezza del passo cambieranno ad ogni giro del ciclo esterno. Supponiamo ad esempio che M1 = 1, M2 = 10 ed M3 = 2. Al primo giro del ciclo esterno lo statement 50 varrà

```
50   FOR B = 1 TO 2 STEP .2
```

(5-24a)

Al secondo giro del ciclo esterno, lo statement 50 varrà invece

```
50   FOR B = 3 TO STEP .6
```

(5-24b)

e così via.

Con i cicli annidati si possono usare gli statements di controllo, che seguono le regole illustrate negli ultimi paragrafi: per *ciascun* ciclo, il controllo può essere trasferito o all'interno stesso del ciclo, o al suo esterno, ma non dall'esterno di un ciclo all'interno di quel ciclo. (L'ultima regola ammette delle eccezioni, ma, in linea generale, è meglio non tenerne conto.)

Rivediamo le regole. Il controllo può essere trasferito da un punto all'altro di un ciclo interno: chiaramente, in questo modo il controllo rimane anche entro l'intervallo del ciclo esterno. Il controllo può passare dal ciclo interno ad uno statement posto al di fuori di esso, ma comunque dentro il ciclo esterno; ancora, il controllo può passare da uno statement posto nel ciclo interno ad uno statement esterno rispetto ad entrambi i cicli. Ma, viceversa, il controllo non può passare da uno statement posto nel ciclo esterno (ma non in quello interno) ad uno statement che faccia parte del ciclo interno; inoltre, il controllo non può passare da uno statement esterno rispetto ad entrambi i cicli a degli statements appartenenti all'uno o all'altro ciclo.

Vediamo ora un altro esempio di cicli annidati. Supponiamo di voler ricavare i valori delle funzioni seguenti:

$$f_1 = x^2 + 2x + 3y + 1$$
$$f_2 = x^2 - 2y + 2x$$

per i seguenti valori di x e di y:

$$x = 0.0,\ 0.1,\ 0.2,\ 0.3,\ \ldots,\ 1.0$$
$$y = 0.0,\ 0.1,\ 0.2,\ 0.3,\ \ldots,\ 1.0$$

Inoltre vogliamo individuare una serie di valori di x e di y per i quali $f_1 - f_2$ raggiunge il valore massimo; vogliamo pure conoscere il valore del massimo $|f_1 - f_2|$.

Questo è realizzato dal programma che segue, mentre in Figura 5-8 è rappresentato il diagramma di flusso.

```
 10  REM CALCOLO DI F1,F2
 15  REM E DEL MAX F1—F2
 20  REM PORRE IL VALORE INIZIALE DI MAX
 30  LET M = 0
 40  PRINT "X","Y","F1","F2"
 50  REM CICLI PER CALCOLARE F1,F2
 60  FOR X = 0 TO 1 STEP .1
 70  FOR Y = 0 TO 1 STEP .1
 80  LET F1 = X↑2+2*X+3*Y+1
 90  LET F2 = X↑2—2*Y+2*X
100  PRINT X,Y,F1,F2
110  REM CALCOLO DI F1—F2
120  LET D = F1—F2
130  REM RENDERE D POSITIVO
140  IF D > = 0 THEN 170
150  LET D = —D
160  REM PORRE M = D SE D> M
170  IF D < = M THEN 210
180  LET M = D
190  LET X1 = X
200  LET Y1 = Y
210  NEXT Y
220  NEXT X
230  PRINT "VAL MAX DI F1—F2 =";M;"X =";X1;"Y =";'Y1
240  END
```

(5-25)
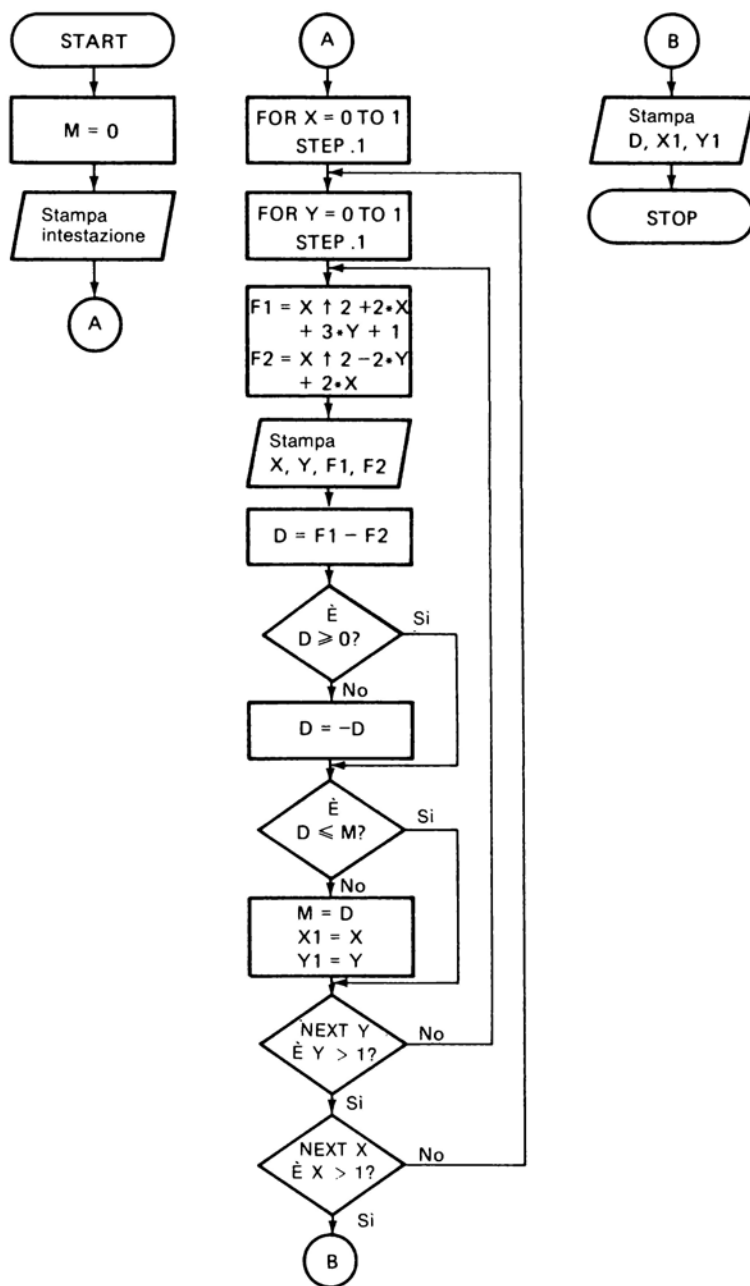
Figura 5.8 — Diagramma di flusso del programma (5-25). Si osservino i simboli di connessione.

Esaminiamone le fasi operative. Nello statement 30 poniamo N = 0: il valore di M alla fine sarà uguale alla massima differenza fra F1 e F2. I valori di x, di y e dei corrispondenti F1 e F2 saranno stampati in una tabella; lo statement 40 stampa le intestazioni della tabella nel modo seguente:

```
X       Y       F1       F2
```

Vengono quindi definiti i due cicli: gli statements 60 e 220 definiscono il ciclo esterno, in cui X varia da 0 ad 1 con passo 0.1, mentre gli statements 70 e 210 definiscono il ciclo interno, nel quale Y varia allo stesso modo. I valori di F1 ed F2 vengono calcolati negli statements 80 e 90, che si trovano nel ciclo interno, ed ogni volta i valori calcolati di F1 ed F2, nonché i valori corrispondenti di X ed Y, sono stampati, secondo l'indicazione dello statement 100. Conclusi i due cicli, la tabella dei valori di F1 ed F2 è completa.

Vediamo ora come si ottiene il massimo di F1—F2. Nello statement 120 si calcola D, cioè la differenza fra F1 ed F2, mentre gli statements 140 e 150 fanno sì che a D (se è negativo) si sostituisca —D, per cui D alla fine rappresenta il valore assoluto di F1—F2. A questo punto D viene confrontato con M. Se M è uguale o maggiore di D, gli statements 180-200 si saltano, ed il ciclo (o i cicli) continua (continuano). Se invece D>M, gli statements 180-200 sono eseguiti, per cui il valore di M viene posto uguale a D, ed X1 e Y1 rispettivamente ai valori attuali di X ed Y: così si aggiornano M, X1 ed Y1 in modo che rappresentino il valore massimo di D ed i valori corrispondenti di X ed Y che sono stati di volta in volta calcolati nel corso dell'esecuzione. Allora, completati i giri dei cicli, M sarà uguale al valore massimo di D, ed X1 e Y1 saranno uguali rispettivamente ai corrispondenti valori di X ed Y. Questi valori vengono quindi stampati, e l'esecuzione termina. Si tenga presente che, se c'è più di una coppia di X ed Y che dà lo stesso valore massimo di D, verrà stampata la prima coppia in cui ci si imbatte. Nel paragrafo 6-2 esamineremo un programma che stampa tutti i valori suddetti.

Abbiamo visto il caso di un ciclo inserito in un altro ciclo. Alcuni sistemi in BASIC ammettono tuttavia l'annidamento di un maggior numero di cicli: ad esempio, un terzo ciclo può essere annidato in un ciclo che, a sua volta, è annidato in un altro ciclo. Anche in questo caso valgono le regole che abbiamo esposto in questo paragrafo: ad esempio ciascun ciclo interno dev'essere trasferito fuori, ma non dentro un ciclo, etc. Il manuale di BASIC che correda il calcolatore utilizzato vi dirà il grado di annidamento consentito.

ESERCIZI

5-1. Illustrate gli effetti del seguente segmento di programma:

```
10  LET N1 = 1
```

```
20  LET N2 = 20
30  FOR X = N1 TO N2

 .

80  NEXT X
```

5-2. Determinate il valore finale di X quando è eseguito il seguente segmento di programma scritto in BASIC:

```
10  LET X = 1
20  FOR I = 1 TO 10
30  X = X+I*X
40  NEXT I
```

Per verificare i risultati ottenuti, scrivete un programma opportuno ed eseguitelo su di un calcolatore.

5-3. Ripetete l'Esercizio 5-2 con la sequente sequenza di programma:

```
10  LET N1 = 1
20  LET N2 = 8*N1
30  FOR I = N1 TO N2
40  X = X+I*X
50  NEXT I
```

5-4. Ripetete l'Esercizio 5-3, ma modificando lo statement 30 nel modo seguente:

```
30  FOR I = N1+2 TO 2*N2
```

5-5. Scrivete un programma che calcoli la somma

$$x = \frac{1}{a} - \frac{1}{a_2} + \frac{1}{a_3} - \frac{1}{a_4} + \ldots - \frac{1}{a_m}$$

in cui a ed m sono introdotte come dati. Disegnate il diagramma di flusso del programma. Verificate il programma eseguendolo su di un calcolatore.

5-6. Ripetete l'Esercizio 5-5, ma facendo in modo che l'elaborazione abbia termine se la grandezza di un qualunque termine della sommatoria diventa minore di d, ove d dev'essere introdotto come dato.

5-7. Scrivete un programma che calcoli la funzione

$$f = x^3 + 2x^2 - 20x - 15$$

per valori di x compresi fra 0 e 10. Variate x di 0.1 per volta. Disegnate il diagramma di flusso del programma. Verificate il programma eseguendolo su di un calcolatore.

5-8. Ripetete l'Esercizio 5-7 con

$$f = -x^3 + 2x^2 + 20x + 15$$

5-9. Illustrate l'uso corretto ed errato degli statements di controllo abbinati ad un ciclo.

5-10. Illustrate gli effetti della seguente sequenza di programma:

```
10  LET N1 = 1.3
20  LET N2 = 18.4
30  LET N3 = 2.6
40  FOR X = N1 TO N2 STEP N3



    .
80  NEXT X
```

5-11. Ripetete l'Esercizio 5-2, ma modificando lo statement 20 nel modo seguente:

```
20  FOR I = 1 TO 10 STEP .5
```

5-12. Ripetete l'Esercizio 5-2, ma con lo statement 20 modificato nel modo seguente:

```
20  FOR I = 1 TO 10 STEP 4.5
```

5-13. Ripetete l'Esercizio 5-3, ma modificando lo statement 30 nel modo seguente:

```
30  FOR I = N1 TO N2 STEP N2/4
```

5-14. Illustrate le fasi operative del seguente programma scritto in BASIC. Qual è il valore finale di A?

```
10  A = 1.0
20  FOR B = 1 TO 5 STEP .5
30  LET A = A+B↑2
40  IF A > 1.72 THEN 60
50  NEXT B
60  PRINT A
70  END
```

Verificate i risultati ottenuti eseguendo il programma su di un calcolatore.

5-15. Modificate il programma dell'Esercizio 5-7 in modo che vengano stampati tutti i valori di f minori di −15. Dovranno essere stampati anche i corrispondenti valori di x. Disegnate il diagramma di flusso del programma. Verificate il programma eseguendolo su di un calcolatore.

5-16. Modificate il programma dell'Esercizio 5-7 in modo che vengano stampati il valore massimo ed il valore minimo di f. Dovranno essere stampati anche i

corrispondenti valori di x. Disegnate il diagramma di flusso del programma. Verificate il programma eseguendolo su di un calcolatore.

5-17. Ripetete l'Esercizio 5-16, ma con riferimento al programma dell'Esercizio 5-8.

5-18. Determinate il valore finale di x quando è eseguito il seguente programma:

```
 10  LET N1 = 2
 20  LET N2 = 9
 30  LET N3 = 0.25
 40  X = 1
 50  FOR A = N1 TO N2 STEP 3*N3
 60  X = X+A
 70  FOR B = N1/2 TO N2/3 STEP N3
 80  X = X+B
 90  NEXT B
100  NEXT A
110  PRINT X
120  END
```

Verificate il risultato ottenuto eseguendo il programma su di un calcolatore.

5-19. Scrivete un programma in BASIC che calcoli e stampi il valore della funzione z, definita qui di seguito. Dovranno essere stampati anche i corrispondenti valori di x ed y. Usate per x valori compresi fra 0 e 15, per y valori compresi fra 0 e 10. Variate x ed y di 0.1 per volta.

$$z = 2y^3x - x^2y + x^2 - y^2 + 20x - 20y + 6$$

Verificate il programma eseguendolo su di un calcolatore.

5-20. Ripetete l'Esercizio 5-19 con la funzione

$$z = -y^3 + x^3 - x^2 + 10\,x - 10\,y - 25$$

5-21. Modificate il programma dell'Esercizio 5-19 in modo che vengano stampati i valori massimo e minimo di z ed i corrispondenti valori di x ed y. Verificate il programma eseguendolo su di un calcolatore.

5-22. Ripetete l'Esercizio 5-21 con il programma dell'Esercizio 5-20.

5-23. Considerate il sistema di due equazioni

$$y = 1 - x^3$$
$$y = x$$

Questo sistema ha una soluzione nell'intervallo $0 \leqslant x \leqslant 1$. Disegnate il diagramma di flusso e scrivete un programma in BASIC che determini una soluzione

con uno scarto massimo di 0.0001. (Si tenga presente che si deve ottenere il valore di x.) Verificate il programma eseguendolo su di un calcolatore.

N.B.: La soluzione deve soddisfare l'equazione $1-x^3-x = 0$. Si può trovare un valore di x variando x di 0.0001 per volta, ponendo attenzione al fatto che $1-x^3-x$ cambierà di segno quando la radice si trova fra due incrementi contigui. (Se la radice coincide esattamente con uno dei valori di campionamento, avremo $1-x^3-x = 0$.) La realizzazione di questo procedimento richiederà molti passaggi, che possono essere ridotti nel modo seguente: assumiamo che x_0 sia la radice; se $x < x_0$, $1-x^3-x$ sarà positivo; se $x > x_0$, $1-x^3-x$ sarà negativo. Allora, provando con 0.5 come valore di x_0, se $1-x^3-x$ è positivo (negativo), 0.5 è troppo piccolo (grande). Se 0.5 è maggiore della radice, provate a dimezzare il valore (0.25); se questo è troppo piccolo (grande), provate con 0.375 (oppure con 0.125). Ripetete questo procedimento con incrementi uguali alla metà di quello usato la volta precedente. Continuate così fino ad ottenere la precisione desiderata.

CAPITOLO 6

I VETTORI

Molto spesso si ha la necessità di scrivere un programma in BASIC che ripeta più volte dei determinati calcoli. Supponiamo ad esempio che in una classe ci siano quaranta studenti, ciascuno dei quali fa quattro esercitazioni. Vogliamo ricavare la media dei loro voti, e poi assegnare a ciascuno studente una lettera-voto, in base alla media. Potremmo scrivere un programma che esegua queste operazioni, e poi rieseguirlo più volte; esiste però una procedura molto più vantaggiosa, della quale parleremo in questo capitolo. Qui esamineremo i *vettori* (*arrays* in inglese) che non sono altro che liste, o tabelle di dati, con le quali si è in grado di semplificare molti aspetti della programmazione. Si ricorre ai vettori non solo quando si devono ripetere dei calcoli su un gran numero di dati, ma anche nel caso di calcoli matematici complessi.

6-1. VETTORI AD UNA DIMENSIONE. VARIABILI CON UN SOLO INDICE

In molti casi è vantaggioso lavorare con variabili con indici; a_1, a_2 ed a_3, ad esempio, sono variabili con un indice. Supponiamo di avere dieci variabili e di volerle sommare. Potremmo scrivere:

$$s = a_1+a_2+a_3+a_4+a_5+a_6+a_7+a_8+a_9+a_{10} \tag{6-1}$$

Ma gl'indici ci permettono di scrivere la sommatoria in modo molto più semplice; ad esempio, l'espressione seguente equivale alla (6-1):

$$s = \sum_{k=1}^{10} a_k \tag{6-2}$$

Il segno Σ indica la sommatoria dei termini a_k per tutti i valori di k da 1 a 10. Usando le variabili con indice, possiamo spesso ridurre le dimensioni del programma e, di conseguenza, renderne più facile la scrittura.

Non si possono introdurre convenientemente gl'indici mediante telescrivente o schede perforate, dato che l'informazione deve giacere tutta su una singola linea; allora, per indicare una variabile con indice, in BASIC si ricorre alla notazione seguente:

A(1)
A(3) (6-3)

Cioè una variabile con indice viene rappresentata come una variabile seguita da una coppia di parentesi che racchiudono un numero intero. In realtà, le parentesi non devono contenere necessariamente un numero intero, in quanto si può usare anche una variabile o un'espressione. Ecco ad esempio delle variabili con indice corrette:

A(I)
B(A1)
C(I+2) (6-4)

Come regola generale, gl'indici non devono essere negativi: se s'impiegano indici negativi, viene stampato un messaggio di errore e l'esecuzione s'interrompe. Alcune versioni di BASIC ammettono l'uso dell'indice 0 (zero), altre non lo ammettono. Il manuale di BASIC del calcolatore utilizzato vi chiarirà questo punto. Se l'indice è un numero decimale, verrà troncato.

Una serie di variabili indicizzate con lo stesso nome è detta *vettore*. Così

A(1),A(2)...A(10) (6-5a)

forma un vettore.

B(1),B(2)...B(10) (6-5b)

è un altro vettore.

Per indicare un vettore si può usare solo un'unica lettera: B2(I) ed AB(J) sono nomi di vettore errati, perché, per denominare il vettore, nel primo si utilizzano una lettera ed un numero, nel secondo due lettere.

Ciascuna variabile indicizzata è detta *elemento* del vettore: ad esempio, nel (6-5b), B(1) è un elemento del vettore. Per il momento supponiamo che tutti i vettori

siano formati da dieci elementi; nel prossimo paragrafo parleremo delle procedure per aumentare o diminuire questo numero.

Scriviamo ora un semplice programma per illustrare l'uso dei vettori. Supponiamo di voler introdurre un gruppo di numeri che varino da 1 a 10, e di volerne calcolare la media: i dati che devono essere introdotti da telescrivente sono N (numero totale degli elementi), seguito dai dati effettivi, uno per linea.

```
 10  REM PROGRAMMA DI MEDIA
 20  PRINT "INTRODURRE IL NUMERO DI ELEMENTI"
 30  INPUT N
 40  LET S = 0
 50  PRINT "INTRODURRE I DATI UNO PER LINEA"
 60  FOR I = 1 TO N
 70  INPUT A(I)
 80  LET S = S+A(I)
 90  NEXT I
100  LET B = S/N
110  PRINT "MEDIA =";B
120  END                                          (6-6)
```

Esaminiamo il programma, il cui diagramma di flusso è in Figura 6-1. Il numero di elementi di cui va calcolata la media viene introdotto nello statement 30; lo statement 50 pone la variabile S uguale a 0: al termine dell'esecuzione, S sarà uguale alla somma di tutti gli elementi di cui bisogna calcolare la media. Lo statement 50 avvisa l'utilizzatore di introdurre i dati, mentre negli statements 60-90 viene definito un ciclo, che sarà eseguito per N volte. Ad ogni giro del ciclo, sarà eseguito lo statement 70, mediante il quale sulla telescrivente comparirà un ? come richiesta d'ingresso di dati: al primo ciclo, questo sarà memorizzato come A(1), al secondo ciclo sarà memorizzato come A(2), e così via. All'esecuzione dello statement 80, S verrà incrementata del valore di A(I) che è stato appena introdotto, così, quando il ciclo avrà compiuto N giri, S sarà la somma di tutte le N variabili.

Terminato il ciclo, viene calcolata la media nello statement 100; la media viene quindi stampata e l'eleborazione ha termine. Una tipica esecuzione del programma sarà:

```
>RUN
MED           9:45          3-MAR-77
INTRODURRE IL NUMERO DI ELEMENTI
? 4
```

```
INTRODURRE I DATI UNO PER LINEA
? 5
? 8
? 7
? 20
MEDIA = 10                                              (6-7)
```
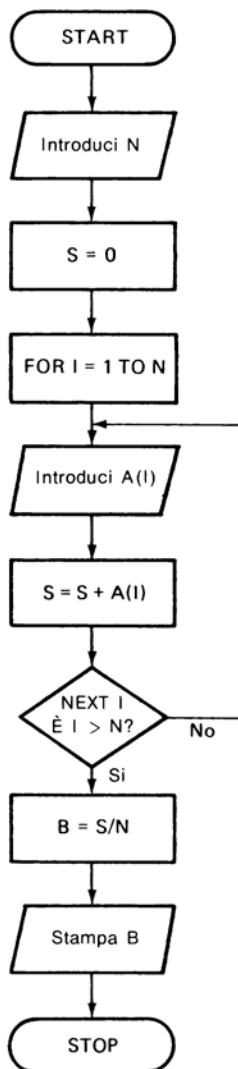
Figura 6.1 — Diagramma di flusso del programma (6-7) che utilizza un vettore per calcolare la media.

6-1-1. Osservazioni sulla denominazione dei vettori

Un vettore ed una differente variabile possono avere lo stesso nome; ad esempio, in uno stesso programma in BASIC, possiamo avere un vettore costituito dagli elementi A(1), A(2), etc., ed un'altra variabile di nome A. Il calcolatore non confonderà le variabili: si noti che A è distinguibile da un vettore per il fatto che non ha parentesi. Anche se il calcolatore non confonderà mai le variabili, questo può capitare spesso all'utilizzatore; pertanto è buona regola di programmazione *non* usare un vettore ed un'altra variabile con lo stesso nome.

6-2. ALTRI ESEMPI DI USO DEI VETTORI AD UNA DIMENSIONE

Nel paragrafo precedente siamo partiti dal presupposto che tutti i vettori ad una dimensione siano costituiti da dieci elementi; in questo paragrafo si parlerà di uno statement che consente di variare il numero di elementi di un vettore. Prima però vediamo come vengono memorizzate le variabili di un programma in BASIC: questo servirà a spiegare perché si abbia bisogno di un siffatto statement.

In fase di compilazione del programma, a ciascuna variabile viene assegnata una locazione di memorizzazione. (La compilazione di un programma prevede in realtà molti passaggi, ma in questa sede ci occuperemo soltanto dell'assegnazione dello spazio di memorizzazione delle variabili.)

Consideriamo dapprima i programmi che non contengono vettori. Durante la compilazione, il programma viene scandito per determinare la locazione di memorizzazione per le variabili, e ad ogni nuova variabile trovata viene riservato uno spazio di memorizzazione. Si noti che, nell'ambito degli statements che si sono esaminati finora, le variabili possono essere introdotte esclusivamente a fronte di statements di READ, INPUT o LET. Più precisamente, gli statements di READ e di INPUT definiscono delle variabili inerenti ai dati d'ingresso, mentre nello statement di LET la nuova variabile viene definita sulla base di variabili e costanti già definite in precedenza. Ora, dato che nella memoria di un calcolatore ci sono specifiche locazioni di memorizzazione riservate per le variabili, il compilatore le assegna alle variabili nell'ordine in cui queste compaiono nel programma. Più esattamente, durante la compilazione di un programma viene generata una lista di locazioni di memorizzazione. Consideriamo ad esempio il programma seguente:

```
10  REM PROGRAMMA CAMPIONE ILLUSTRANTE
15  REM LA MEMORIZZAZIONE
20  INPUT A,B,X2
30  LET D1 = A+B*X2
40  LET A1 = A1*X2
50  LET D1 = A1*X2
60  PRINT D1,A1
99  END                                      (6-8)
```

La lista delle variabili corrispondenti alle locazioni di memorizzazione sarà:

A
B
X2
D1
A1

Esaminiamo l'intero processo. Per primo è scandito lo statement di INPUT, in cui compaiono le tre variabili A, B ed X2, che diventano pertanto i primi tre termini della lista. Il successivo statement è il 30: qui compare D1 alla sinistra del segno uguale, e quindi D1 è aggiunta alla lista. Allo stesso modo, quando viene esaminato lo statement 40, A1 viene aggiunta alla lista. Quando è scandito lo statement 50, si ha un caso diverso: la variabile D1 compare alla sinistra del segno uguale nello statement di LET, ma, d'altra parte, per D1 è già stata riservata una locazione di memorizzazione, e di conseguenza non ne è riservata un'altra. Allora, riassumendo, la procedura generale è che ad ogni variabile è riservata una locazione di memorizzazione nell'ordine in cui le variabili stesse compaiono nel programma; comunque, anche se una variabile compare in uno statement di INPUT o di READ, o, alla sinistra del segno uguale, in uno statement di LET, non le sarà riservata una locazione di memorizzazione se ne è già stata riservata una per una variabile dello stesso nome.

Venendo ai vettori, il calcolatore non è in grado di determinare la lunghezza di un vettore in fase di compilazione. Consideriamo ad esempio il programma

```
10   LET S = 0
20   INPUT A,B
30   FOR I = 1 TO A
40   INPUT E(I)
50   LET S = S+E(I)
60   NEXT I
70   LET G = S+B
80   PRINT S,G
90   END                                        (6-9)
```

Quando il programma viene compilato, S, A e B possono essere determinate come variabili, e può essere loro riservata un'area di memorizzazione. Ma vediamo ora il vettore E(I). Il compilatore non sa di quanti elementi esso è composto: in primo luogo, questo dipende dal valore di A, che viene introdotto in fase di esecuzione, e non di compilazione; poi, quand'anche lo statement 30 fosse scritto in questo modo:

```
30   FOR I = 1 TO 8
```

il compilatore, per la sua struttura, non sarebbe in grado di stabilire esattamente che sono richiesti otto elementi. Inoltre un elemento di un vettore potrebbe essere definito in modo diverso che con un ciclo, per cui, quando tale elemento è esaminato, il compilatore non "sa" quanto spazio di memorizzazione riservare. Per questa ragione si ricorre alla procedura seguente: ogni volta che si trova un nuovo vettore, gli vengono assegnati dieci spazi di memorizzazione. (Vedremo poi che questa procedura è passibile di varianti.) Quindi la lista delle variabili memorizzate per il programma (6-9) sarà:

S,A,B,E(1),E(2),E(3),E(4),E(5),E(6),E(7),E(8),E(9),E(10),G

Abbiamo supposto che gl'indici vadano da 1 a 10, per quanto alcuni compilatori ammettano anche l'indice 0: in questo caso saranno riservati undici spazi di memorizzazione, in cui, con riferimento all'esempio precedente, E(0) sarebbe memorizzato prima di E(1) e dopo B.

6-2-1. Lo statement di DIM

Molto spesso si ha bisogno di lavorare con vettori formati da più di dieci elementi. A tal fine dobbiamo includere uno statement particolare, lo statement di DIM, che indica al compilatore la quantità di spazio di memorizzazione da riservare. La forma dello statement di DIM è:

```
10  DIM E(30)                                        (6-10)
```

Questo in particolare segnala al compilatore che per il vettore E(I) vanno riservate in memoria trenta locazioni, che diventano trentuno nei casi in cui è ammesso l'indice 0. Diciamo che il vettore E(I) è stato *dimensionato* nello statement (6-10).

In uno statement di DIM si può dimensionare più di un vettore. Ad esempio potremo avere:

```
10  DIM E(30),A(50),B(4)                             (6-11)
```

Qui il vettore E è dimensionato per trenta elementi, il vettore A per cinquanta elementi, ed il vettore B per quattro elementi. (Nei casi in cui è ammesso l'indice zero, ciascun vettore avrà un elemento in più.)

La forma generale dello statement di DIM è: numero di linea, la parola-chiave DIM, seguita da una lista di nomi di vettori con le relative dimensioni poste fra parentesi. Si noti che i vettori sono separati da virgole, e che non ci sono virgole né alla fine della lista né dopo la parola DIM.

In (6-11) abbiamo stabilito che il vettore B debba avere quattro elementi. Questo

sembrerà forse superfluo, dal momento che un vettore non definito avrà automaticamente riservati dieci spazi in memoria; ma si pensi che, se noi sappiamo che occorrono meno di dieci spazi, il dimensionare il vettore ci permette di non sprecare locazioni di memoria.

Si tenga presente che per dimensionare un vettore *non si possono usare* variabili o espressioni. Ad esempio,

```
10  DIM A(K)
```

è *errato*, perché il valore di K non può essere determinato dal compilatore. Le sole quantità che si possono usare per dimensionare un vettore sono i numeri interi privi di segno.

In un programma scritto in BASIC può comparire più di uno statement di DIM, ma un vettore può comparire una sola volta: in altre parole, un vettore *non può* venir dimensionato in più di uno statement, né due volte nello stesso statement di DIM. Gli statements di DIM possono trovarsi in qualsiasi punto del programma, ma è buona regola di programmazione collocarli tutti all'inizio; in tal modo è facile individuarli ed accertarsi che tutti i vettori siano stati dimensionati in modo corretto.

Vediamo un altro esempio di programma in BASIC che utilizza un vettore. Supponiamo che in una classe ci siano K studenti, ove K è uguale o minore di 100. Ciascuno studente fa quattro esercitazioni. Vogliamo calcolare la media dei voti riportati da ciascuno studente. Identificheremo gli studenti con un numero d'identificazione (il numero ID). *Non* è necessario che i voti riportati dagli studenti siano introdotti nell'ordine corrispondente ai relativi numeri di identificazione. Quando il programma è eseguito, verranno stampati il numero di identificazione e la media di ciascuno studente, in un elenco che seguirà l'ordine sequenziale (primo lo studente n. 1, secondo lo studente n. 2, etc.); saranno inoltre stampate la media più bassa e quella più alta con il corrispondente numero dello studente (o i corrispondenti numeri degli studenti). Il programma è il seguente:

```
 10  REM PROGRAMMA DI MEDIA
 20  DIM A(100)
 30  PRINT "INTRODURRE IL NUMERO TOTALE",
 35  PRINT "DEGLI STUDENTI";
 40  INPUT N
 50  LET B = 0
 60  LET S = 100
 70  REM INTRODUZIONE VOTI E
 75  REM CALCOLO MEDIA
 80  PRINT "INTRODURRE IL NUMERO ED I VOTI",
 85  PRINT "DELLO STUDENTE";
 90  FOR I = 1 TO N
100  INPUT K,G1,G2,G3,G4
```

```
110  LET A(K) = (G1+G2+G3+G4)/4
120  REM TROVA LA MEDIA PIU' ALTA
125  REM E QUELLA PIU' BASSA
130  IF B > = A(K) THEN 150
140  LET B = A(K)
150  IF S < = A(K) THEN 170
160  LET S = A(K)
170  NEXT I
180  REM RICAVA MEDIE E NUMERI
185  REM DEGLI STUDENTI IN ORDINE
190  PRINT "NUM. ID","MEDIA"
200  FOR T = 1 TO N
210  PRINT T,A(T)
220  NEXT T
230  REM STAMPA NUM. ID STUDENTI CON
235  REM LE MEDIE PIU' ALTE E PIU' BASSE
240  PRINT "MEDIA MASSIMA =";B
250  PRINT "I NUMERI DI ID DEGLI STUDENTI",
255  PRINT "RELATIVI SONO"
260  FOR V = 1 TO N
270  IF A(V)< > B THEN 290
280  PRINT V
290  NEXT V
300  PRINT "MEDIA MINIMA =";S
310  PRINT "I NUMERI DI ID DEGLI STUDENTI",
315  PRINT "RELATIVI SONO"
320  FOR W = 1 TO N
330  IF A(W)< > S THEN 350
340  PRINT W
350  NEXT W
999  END                                              (6-12)
```

In Figura 6-2 è rappresentato il diagramma di flusso del programma. Vediamone le fasi operative. Per memorizzare le medie degli studenti sarà necessario un vettore: dal momento che possono esserci 100 studenti al massimo, dimensioniamo il vettore come A(100). A questo punto introduciamo il numero totale degli studenti. (Non dimentichiamo che non si può cambiare la dimensione del vettore durante l'esecuzione.) Negli statements 50 e 60 poniamo il valore iniziale di B a 0 e il valore iniziale di S a 100: questi valori alla fine saranno uguali alla media più alta ed a quella più bassa. A questo punto, uno statement di avviso indica all'operatore di introdurre il numero dello studente ed il voto. Negli statements 90-170 viene definito un ciclo, che compirà N giri: ogni volta che si arriva allo statement 100, sulla telescrivente compare un ?, e vengono introdotti il numero d'identificazione dello studente ed i
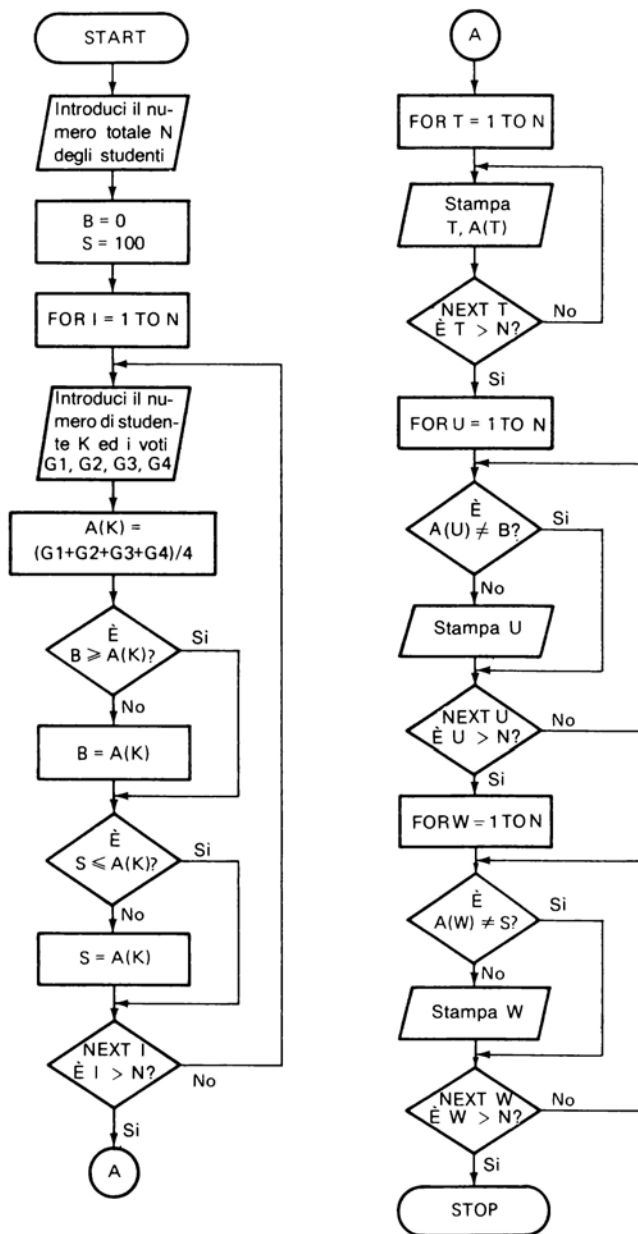
Figura 6.2 — Diagramma di flusso del programma (6-12) che calcola la media dei voti e ricava il numero ID dello studente con la media più alta e di quello con la media più bassa.

quattro voti relativi. (Non è necessario che i dati relativi agli studenti siano introdotti secondo l'ordine definito dai numeri ID.) Nella linea 110 viene quindi calcolata la media, che è poi memorizzata nel vettore A(K): si noti che A(1) è la media dello studente numero 1, A(2) è la media dello studente numero 2, e così via.

Completati i giri del ciclo, B risulterà posto dalle linee 130-150 uguale alla media più alta (con un procedimento sostanzialmente identico a quello del programma (5-25)), mentre S risulterà posto dalle linee 150-170 uguale alla media più bassa. Si noti che partiamo da S = 100, che è la più alta media possibile, avendo supposto che la media più bassa sia minore di 100: in realtà, potremmo sostituire S = 100 con S = 101, per evitare i problemi che si presenterebbero se tutti gli studenti avessero la media del 100.

Il ciclo che va dallo statement 200 al 220 fa stampare il numero ID e la media di ogni studente: si noti che, anche se i voti riportati da uno studente vengono introdotti non rispettando l'ordine sequenziale dei numeri ID, la tabella delle medie viene stampata seguendo l'ordine sequenziale dato dai numeri ID. Nel ciclo che va dallo statement 260 allo statement 290 sono stampati i numeri ID degli studenti che hanno riportato il massimo della media: lo statement di IF-THEN numero 270 verifica se A(V) è diverso da B e, in caso affermativo (A(V) ≠ B), si salta lo statement di PRINT; se invece A(V) = B, l'elemento A(N) in questione ha valore uguale al valore massimo, per cui viene stampato il corrispondente V (numero ID dello studente). Completati i giri del ciclo, saranno stampati i numeri ID di tutti gli studenti che hanno riportato una media uguale alla media massima. Analogamente, gli statements 300-350 faranno stampare i numeri ID degli studenti che hanno riportato la media minima. Si noti che la procedura è diversa da quella del programma (5-25), nel quale veniva stampato un solo valore massimo (di X1 e di Y1, che corrispondono ai numeri ID).

6-3. VETTORI A DUE DIMENSIONI

Abbiamo considerato finora vettori i cui elementi hanno un solo indice, essendo questo tipo di vettore utile per rappresentare liste o variabili con un solo indice. Spesso occorre lavorare con vettori costituiti da elementi con due indici: questi sono detti vettori *a due dimensioni*, e sono vantaggiosi per rappresentare tabelle o variabili con due indici. Vediamo una tabella di questo tipo. Nell'ultimo paragrafo abbiamo esaminato un programma che calcolava la media dei voti riportati dagli studenti in quattro esercitazioni, e ci siamo serviti di un vettore ad una dimensione per memorizzare le medie. Supponiamo ora di voler memorizzare, oltre alle medie, anche tutti i voti delle esercitazioni. Sarà allora necessaria una tabella, che avrà ad esempio questa forma:

Numero di identificazione degli studenti	Prova 1	Prova 2	Prova 3	Prova 4	Media
1	100	100	100	100	100
2	80	60	70	70	70
3	100	80	85	85	85
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

(6-13)

Non è necessario che il numero d'identificazione dello studente faccia parte del vettore, perché coincide con la riga del vettore stesso: il medesimo principio è stato utilizzato nel paragrafo 6-2, nel quale il numero ID dello studente era dato dalla riga del vettore. Vediamo un altro esempio in cui è utile ricorrere a vettori a due dimensioni.

Spesso, in matematica, si opera su numeri con due indici, come ad esempio in

$$f_1 = a_{11}x_1 + a_{12}x_2 \quad (6\text{-}14a)$$

$$f_2 = a_{21}x_1 + a_{22}x_2 \quad (6\text{-}14b)$$

in cui è vantaggioso rappresentare le a con un vettore a due dimensioni. In BASIC, dal momento che si lavora con telescriventi, lettori di schede, etc., tutti i simboli sono posti sulla medesima linea, e quindi un elemento di un vettore con due indici sarà rappresentato in questo modo: A(1,2), B(1,3). Per i vettori con due indici valgono sostanzialmente le stesse regole che si usano per quelli con un solo indice, tranne che in questo caso fra parentesi compaiono due numeri interi separati da una virgola. In realtà, al posto dei numeri interi possiamo anche avere costanti o espressioni. Ad esempio, i seguenti elementi di vettori sono leciti:

```
A(I,J)
B(3,K*B)
```

Se il valore della variabile o dell'espressione è un numero non intero, viene troncato (in alcune versioni di BASIC viene arrotondato). La variabile, o l'espressione, non può essere negativa; nel caso che lo sia, è generato un messaggio di errore. Alcuni compilatori ammettono l'uso degl'indici zero, altri non lo ammettono. Si tenga presente che tutte queste regole sono sostanzialmente le stesse viste per i vettori ad una dimensione.

In assenza di uno statement di DIM, si dà per scontato che in BASIC i vettori ad una dimensione siano costituiti da dieci elementi; allo stesso modo, i vettori a due dimensioni sono automaticamente dimensionati a dieci righe e dieci colonne, e, nei

casi in cui sono ammessi gl'indici zero, avremo undici righe e undici colonne. Si può modificare il numero di righe e di colonne di un vettore a due dimensioni mediante uno statement di DIM. Ad esempio

```
10  DIM A(20,30)                                                    (6-15)
```

farà sì che il vettore A abbia venti righe e trenta colonne. In un unico statement di DIM si può dimensionare più di un vettore. Ad esempio,

```
10  DIM A(20,30),B(15),C(100),D(4,6)                                (6-16)
```

Qui per il vettore A parleremo di vettore 20 × 30 (20 righe per 30 colonne), il vettore B è un vettore ad una dimensione di quindici righe, etc. Si tenga presente che, nei casi in cui sono ammessi gl'indici zero, A sarà 21 × 31, B avrà 16 righe, e così via. Si ricordi che un vettore dev'essere dimensionato se ha *o* più di dieci righe *o* più di dieci colonne o anche se rientra in *entrambi* i casi (e s'intenda undici, al posto di dieci, se sono ammessi gl'indici zero).

Si tenga presente che convenzionalmente si considera il primo indice di un vettore come quello che rappresenta la riga, il secondo come quello che rappresenta la colonna: ad esempio A(2,3) sarà l'elemento del vettore A posto sulla seconda riga della terza colonna.

Spieghiamo l'uso dei vettori a due dimensioni con il programma seguente: supponiamo di avere una classe composta da un massimo di 100 studenti, ciascuno dei quali fa tre esercitazioni. Gli studenti sono contraddistinti dai numeri ID. Il numero ID ed i tre voti riportati da ciascuno studente devono essere introdotti su un'unica linea, mentre non è indispensabile che i dati entrino nell'ordine definito dai numeri ID. Lo scopo del programma è calcolare la media di ciascuno studente e, una volta introdotti i dati, stampare una tabella le cui voci saranno il numero d'identificazione, il voto di ciascuna esercitazione e la media. Sarà poi stampata un'altra tabella di due colonne: il numero ID dello studente e la media. Questa tabella sarà disposta secondo l'ordine delle medie, nel senso che per prima sarà stampata la media più alta, poi quella che vien dopo, e così via. Questo è il programma:

```
10  REM PROGRAMMA CHE CALCOLA LE MEDIE
15  REM E LE ELENCA
20  DIM A(100,4)
30  REM INTRODUZIONE DATI
40  PRINT "INTRODURRE IL NUMERO TOTALE",
45  PRINT "DEGLI STUDENTI"
50  INPUT N
60  PRINT "INTRODURRE IL NUM. ID",
65  PRINT "E TRE VOTI"
```

```
 70  FOR I = 1 TO N
 80  INPUT K,A(K,1),A(K,2),A(K,3)
 90  LET A(K,4) = (A(K,1)+A(K,2)+A(K,3))/3
100  NEXT I
110  REM STAMPA TABELLA
120  PRINT "NUM. DI ID","VOTO 1","VOTO 2","VOTO 3","MED"
130  FOR J = 1 TO N
140  PRINT J,A(J,1),A(J,2),A(J,3),A(J,4)
150  NEXT J
160  REM ORDINA IN ORDINE DECRESCENTE
170  PRINT "NUM. ID","MED"
180  FOR T = 1 TO N
190  LET B = 0
200  FOR R = 1 TO N
210  IF A(R,4)<= B THEN 240
220  LET B = A(R,4)
230  LET R1 = R
240  NEXT R
250  PRINT R1,A(R1,4)
260  LET A(R1,4) = 0
270  NEXT T
999  END                                                    (6-17)
```

In Figura 6-3 è rappresentato il diagramma di flusso del programma. Vediamone le fasi operative. Il vettore è dimensionato nello statement 20, mentre per mezzo dello statement 50 è introdotto il numero effettivo degli studenti. Le linee da 70 a 100 costituiscono un ciclo che provvede all'introduzione dei dati ed al calcolo della media. Esaminiamo l'introduzione dei dati. Lo statement 80 ha la forma:

```
80  INPUT K,A(K,1),A(K,2),A(K,3)                            (6-18)
```

dove prima è introdotto K, poi A(K,1), A(K,2) ed A(K,3). Quindi, se sono introdotti 6, 50, 100 e 90, gli elementi del vettore saranno A(6,1) = 50, A(6,2) = 100 ed A(6,3) = 90. Si noti che *il valore di K dev'essere introdotto per primo.* Di conseguenza

```
INPUT A(K,1),A(K,2),A(K,3),K
```

sarebbe *sbagliato*, perché vorrebbe dire pretendere di introdurre A(K,1), etc. prima di dare il valore di K: non si deve procedere in questo modo perché, quando il valore di K non è noto (o è memorizzato il valore sbagliato), A(K,1), etc. non sono definiti. In altre parole, "il calcolatore deve conoscere" il nome della locazione di memoria, e quindi il valore numerico di K dev'essere noto.
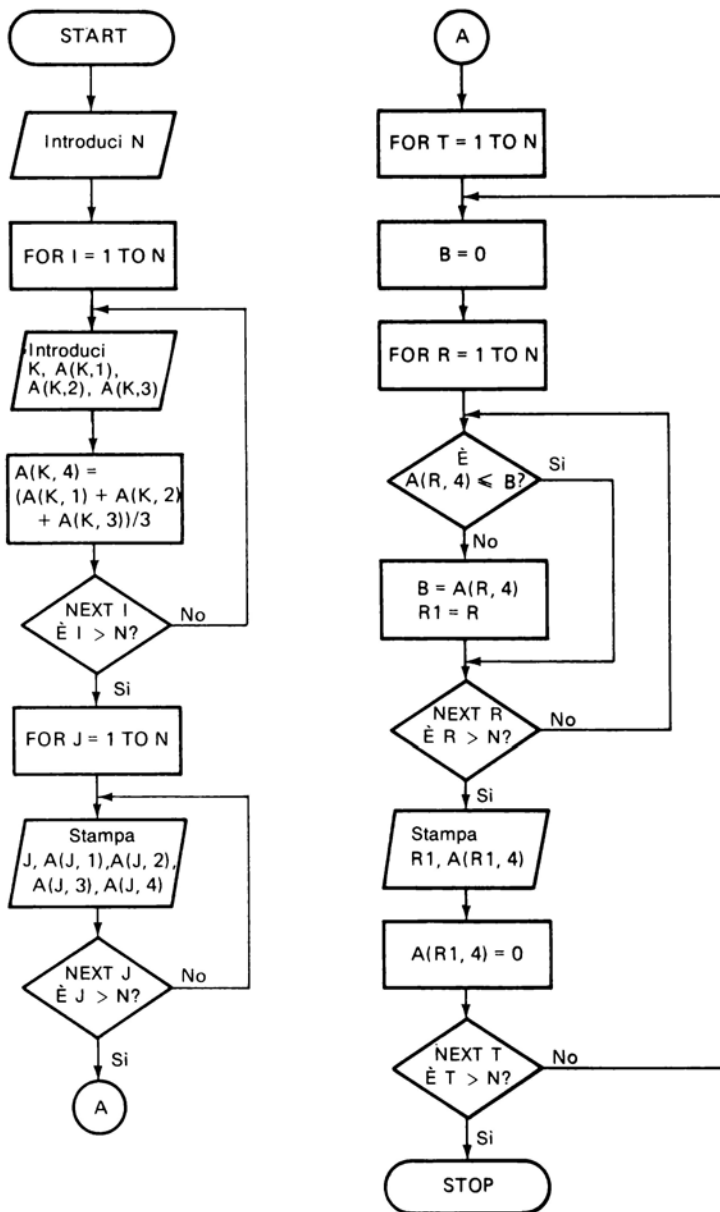
Figura 6.3 — Diagramma di flusso del programma (6-17) che calcola la media dei voti e stampa una tabella in cui le medie sono elencate in ordine decrescente.

Nelle linee 130-150 è definito un ciclo che porta alla stampa della tabella contenente il numero ID, i voti e la media; questa tabella seguirà l'ordine dei numeri ID degli studenti.

Per stampare ora la lista dei numeri ID e delle medie, che segua l'ordine definito dalle medie, useremo una procedura analoga a quella del programma (6-12), nel quale venivano stampati i numeri ID degli studenti che avevano riportato la media più alta. Vediamola. Abbiamo ora due cicli: quello esterno è definito negli statements 180 e 240. Il ciclo interno è sostanzialmente lo stesso di quello corrispondente nel programma (6-12). In primo luogo si pone B = 0; quindi, eseguito il primo giro del ciclo interno, B sarà uguale alla media più alta, ed R1 al numero ID dello studente che ha riportato quella media: si noti che, se la media più alta è stata ottenuta da due o più studenti, il valore di R1 corrisponderà a quello che ha il più basso numero ID. Dopo che il controllo lascia il ciclo interno, vengono stampati R1 ed A(R1,4), che corrispondono rispettivamente al numero ID dello studente che ha riportato la media più alta ed alla media più alta medesima. A questo punto si pone A(R1,4) uguale a 0, ed immediatamente dopo riparte il ciclo esterno: B è posto di nuovo uguale a 0; poi è attivato il ciclo interno, e quindi la procedura già vista si ripete. Come risultato, viene trovata ancora una volta la media più alta, ma la lista delle medie è cambiata, in quanto nel giro precedente era la media più alta ad essere posta uguale a 0 nella linea 260: quindi, all'esecuzione del ciclo interno, si individua la seconda media in graduatoria, che sarà perciò stampata insieme con il corrispondente numero ID. Se due o più studenti hanno riportato la stessa media più alta, sarà tolto il secondo. Andando avanti, ad ogni giro del ciclo esterno (e corrispondente esecuzione completa del ciclo interno) sarà stampato un altro numero ID con la relativa media. Al termine dell'esecuzione del ciclo esterno, avremo stampati nel giusto ordine tutti i numeri ID con le relative medie.

6-3-1. Ancora sull'introduzione dei dati

Si è visto (v. lo statement (6-18)) come la riga di un vettore possa essere specificata tramite dati introdotti da tastiera. In realtà, possiamo determinare sia la riga che la colonna in cui memorizzare un particolare valore di dato. Supponiamo ad esempio di avere un vettore con M righe ed N colonne, e di voler introdurre negli elementi del vettore dei dati in un ordine qualunque. Potremo utilizzare il seguente segmento di programma:

```
60  FOR I = 1 TO M
70  FOR J = 1 TO N
80  INPUT K,L,A(K,L)
```

```
 90  NEXT J
100  NEXT I
```

(6-19)

Esaminiamo lo statement 80: sono *dapprima* introdotti K ed L, e *poi* A(K,L). Cioè, nel momento in cui è introdotto A(K,L), i valori di K e di L sono già noti. Lo statement seguente è esattamente equivalente allo statement 80:

```
80  INPUT L,K,A(K,L)
```
(6-20)

Se ad esempio in risposta al punto interrogativo di avviso abbiamo

```
? 4,6,75
```

allora è fatta la seguente assegnazione: A(6,4) = 75. Si noti che è A(6,4) e non A(4,6), anche se 4 è introdotto per primo: infatti 4 è memorizzato nella locazione L, e 6 nella locazione K. Perciò quando lavoriamo con A (K,L), il valore memorizzato nella locazione K viene utilizzato come primo indice, mentre il valore memorizzato nella locazione L viene utilizzato come secondo indice.

Si tenga presente che K ed L devono venire introdotti prima di A(K,L); quindi gli statements

```
10  INPUT A(K,L),K,L
```
(6-21a)

```
10  INPUT K,A(K,L),L
```
(6-21b)

e così via sono *errati*, perché A(K,L) non è definita come una ben specifica locazione di memoria finché non siano introdotti K ed L. Pertanto, se tentassimo di usare la (6-21a) o la (6-21b), la locazione di memoria A(K,L) non sarebbe definita (o lo sarebbe in modo errato) al momento dell'introduzione dei dati relativi ad A(K,L). Di conseguenza, i dati non sarebbero memorizzati correttamente.

Abbiamo esposto questi concetti in relazione ai vettori a due dimensioni; ma è chiaro che essi valgono anche per i vettori ad una dimensione.

ESERCIZI

6-1. Considerate il segmento di programma

```
      .
20  LET I = 2
30  LET J = I*2
40  LET A(I↑J) = 4.3
50  LET A(I+J) = 6.7
```

Qual è il valore di A(6)? Verificate il risultato ottenuto scrivendo un opportuno programma in BASIC ed eseguendolo su di un calcolatore.

6-2. Scrivete un programma che accetti N numeri e ne calcoli il prodotto (N è un numero minore di 10 e dev'essere introdotto come dato).

6-3. Quale valore di B verrà stampato all'esecuzione del programma seguente?

```
10  FOR I = 1 TO 5
20  LET A(I) = I
30  NEXT I
40  B = 0
50  FOR J = 1 TO T STEP .5
60  B = B+A(J)
70  NEXT J
80  PRINT B
90  END
```

Verificate il risultato ottenuto eseguendo il programma su di un calcolatore.

6-4. Ilustrate la memorizzazione delle variabili.

6-5. A che cosa servono gli statements di DIM?

6-6. Qual è il significato del seguente statement BASIC?

```
10  DIM A(15),B(6),C(150)
```

6-7. Modificate il programma dell'esempio (6-12) nel modo seguente: la differenza fra le medie massima e minima degli studenti va divisa in cinque intervalli uguali, corrispondenti alle lettere-voto A, B, C, D ed F. Quando vengono stampati i numeri ID e le medie degli studenti, con essi andrà stampata anche la lettera - voto appropriata.

6-8. Scrivete un programma in BASIC che calcoli la funzione

$$F = a_5x^5 + a_4x^4 + \ldots + a_1x + a_0$$

in cui le a devono essere memorizzate in un vettore. I valori di a_i e di x vanno introdotti come dati. Disegnate il diagramma di flusso del programma. Verificate il risultato ottenuto eseguendo il programma su di un calcolatore.

6-9. Modificate il programma dell'Esercizio 6-8 in modo che si possa calcolare la funzione

$$F = a_nx^n + a_{n-1}x^{n-1} + \ldots + a_1x + a_0$$

Anche il valore di n dev'essere introdotto come dato. Come indicazione generale, $n \leqslant 150$.

6-10. Illustrate gli effetti dello statement seguente:

```
20  INPUT A,B,C(A+1,B)
```

6-11. Perché lo statement seguente è in assoluto errato?

```
20  INPUT A,C(A,B),B
```

6-12. Illustrate l'uso dei vettori a due dimensioni.

6-13. Descrivete le fasi operative del programma seguente:

```
 10  DIM A(20,30)
 20  FOR I = 1 TO 20
 30  FOR J = 1 TO 29
 40  LET A(I,J) = I+J*2
 50  NEXT J
 60  NEXT I
 70  FOR K = 1 TO 20
 80  A(K,30) = 0
 90  NEXT K
100  FOR M = 1 TO 20
110  FOR P = 1 TO 29
120  LET A(M,30) = A(M,30)+A(M,P)
130  NEXT P
140  PRINT M,A(M,P)
150  NEXT M
160  END
```

Verificate il risultato ottenuto eseguendo il programma su di un calcolatore.

6-14. Modificate il programma (6-12) in modo che le medie vengano stampate in ordine crescente, invece che decrescente.

6-15. Modificate il programma (6-17) in modo che l'utilizzatore possa introdurre gl'intervalli di medie da utilizzare per le lettere-voto A, B, C, D ed F. Poi vanno stampati il numero ID e la lettera-voto dello studente. Disegnate il diagramma di flusso del programma. Verificate il programma eseguendolo su di un calcolatore.

6-16. Modificate il programma dell'Esercizio 6-15 in modo che venga stampato il posto in graduatoria dello studente all'interno della classe.

6-17. Scrivete un programma utilizzante dei vettori per risolvere il seguente sistema di equazioni, nel quale a_{ij} e y_{yj} siano valori noti.

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = y_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = y_2$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = y_3$$

Disegnate il diagramma di flusso del programma. Verificate il programma eseguendolo su di un calcolatore.

N.B.: Supponiamo di avere le seguenti equazioni:

$$3x_1 + x_2 - x_3 = 2$$
$$x_1 + 3x_2 + x_3 = 10$$
$$x_1 + 2x_2 + 5x_3 = 20$$

Moltiplichiamo la terza equazione per −3, e sommiamola alla prima; poi moltiplichiamo la terza equazione per −1 e sommiamola alla seconda. Avremo

$$0x_1 - 5x_2 - 16x_3 = -58$$
$$0x_1 + x_2 - 4x_3 = -10$$
$$x_1 + 2x_2 + 5x_3 = 20$$

Ora, se moltiplichiamo la seconda equazione per 5, e quindi la sommiamo alla prima, avremo:

$$0x_1 + 0x_2 - 36x_3 = 108$$
$$0x_1 + x_2 - 4x_3 = -10$$
$$x_1 + 2x_2 + 5x_3 = 20$$

Dividiamo ora la prima equazione per −36. Otterremo

$$0x_1 + 0x_2 + x_3 = 3$$
$$0x_1 + x_2 - 4x_3 = -10$$
$$x_1 + 2x_2 + 5x_3 = 20$$

Ripetendo il procedimento, otterremo infine

$$0x_1 + 0x_2 + x_3 = 3$$
$$0x_1 + x_2 + 0x_3 = 2$$
$$x_1 + 0x_2 + 0x_3 = 1$$

Quindi

$$x_1 = 1$$
$$x_2 = 2$$
$$x_3 = 3$$

e le equazioni sono risolte. Il procedimento che abbiamo usato è detto metodo di riduzione di Gauss.

6-18. Generalizzate il programma dell'Esercizio 6-17 in modo da risolvere un sistema di equazioni ad n incognite. Disegnate il diagramma di flusso del programma. (Si tenga presente che la soluzione di sistemi di equazioni di grado superiore può portare a risultati inesatti, a meno di usare dei procedimenti particolari che vanno oltre l'assunto di questo libro.)

CAPITOLO 7

I SOTTOPROGRAMMI

In programmi complessi, capita spesso di dover ripetere più volte una stessa sequenza di calcoli. Chiaramente si può raggiungere questo scopo ripetendo il segmento di programma interessato con nomi diversi delle variabili, ma questo allunga il programma e richiede del lavoro in più da parte del programmatore. In realtà, non siamo costretti a procedere in questo modo: possiamo invece scrivere il segmento di programma che si ripete, una sola volta, come programma particolare, detto *sottoprogramma*. Uno stesso sottoprogramma può essere utilizzato più volte con variabili diverse. Il programma che utilizza il sottoprogramma è detto *programma principale*. In questo capitolo esamineremo per l'appunto i sottoprogrammi.

In molti casi i sottoprogrammi sono scritti dal programmatore ed inseriti nel programma principale; ma ci sono dei sottoprogrammi che possono essere utilizzati molto frequentemente in un gran numero di programmi. In questo caso, anche se potrebbero essere scritti da un qualunque programmatore, si preferisce, per risparmiare lavoro, includerli nel linguaggio BASIC: questi sottoprogrammi vengono detti *sottoprogrammi incorporati*, o *di libreria*, in quanto non devono essere scritti dal programmatore, e si possono utilizzare in qualsiasi programma BASIC.

Ci sono due categorie di sottoprogrammi: le *funzioni* e le *subroutines*, e in questo capitolo se ne parlerà insieme ai sottoprogrammi di libreria.

7-1. FUNZIONI SU UNA SOLA LINEA

Capita molto spesso che, nei programmi complessi, un calcolo debba essere ripetuto molte volte. L'uso del sottoprogramma funzione evita al programmatore di scrivere più volte questi calcoli. In questo paragrafo esamineremo semplici funzioni che si possono scrivere su una sola linea; nel prossimo vedremo invece funzioni più complesse la cui scrittura richiede più di una linea.

Illustriamo l'uso di una funzione. Supponiamo di voler sommare i quadrati di tre quantità, e poi elevarne al quadrato la somma. Possiamo definire una funzione su una sola linea in questo modo:

```
10  DEF FNA(X,Y,Z) = (X↑2+Y↑2+Z↑2)↑2                    (7-1)
```

La forma è: numero di statement, la parola-chiave DEF, seguita da tre lettere che danno il nome della funzione (le prime due *devono* essere FN), quindi una lista di variabili poste fra parentesi e separate da virgole, e infine il segno uguale. L'operazione che si vuol eseguire è scritta alla destra del segno uguale. Le variabili che compaiono nello statement di definizione di funzione sono dette *variabili fittizie*. Si noti che dal calcolo della funzione si ottiene un unico numero. Vediamo ora l'uso della funzione:

```
10  REM PROGRAMMA CON UNA FUNZIONE
20  DEF FNA(X,Y,Z) = (X↑2+Y↑2+Z↑2)↑2
30  INPUT A,B,C,D
40  LET Y1 = FNA(A,B,C)
50  LET Y2 = FNA(B,C,D)+2*A
60  LET Y3 = FNA(Y1,Y2,Y2)
70  PRINT Y3
99  END                                                (7-2)
```

Esaminiamo le fasi operative del programma. Lo statement 20 definisce la funzione illustrata prima: si noti che, durante l'esecuzione del programma, questo statement non viene eseguito nell'ordine, ma solo quando è richiesto dal programma principale. Poi lo statement 30 fa sì che quattro numeri vengano introdotti ed assegnati alle locazioni di memoria A, B, C e D. Nello statement 40 è richiesta la funzione. La forma dello statement di richiesta è: nome della funzione seguito dalla lista di variabili, il tutto *alla destra* del segno di uguale. Il nome della funzione dev'essere identico al nome della funzione definita, e la lista delle variabili deve coincidere esattamente con quella che compare nello statement di definizione di funzione, nel senso che, se nello statement di definizione compaiono tre variabili, nello statement di richiesta dovranno esserci esattamente tre variabili. Le variabili che compaiono nello statement di richiesta *devono* essere già state definite *prima* dell'esecuzione dello statement di richiesta: a questo punto i valori delle variabili presenti nello statement di richiesta vengono assegnati alle variabili fittizie della funzione, ed il calcolo è eseguito. Ad esempio, l'esecuzione dello statement 40 dà come risultato lo stesso valore numerico che si otterrebbe scrivendo

```
40  LET Y1 = (A↑2+B↑2+C↑2)↑2                           (7-3)
```

Si osservi comunque che, anche se la linea (7-3) e lo statement 40 danno come risultato gli stessi valori numerici, le rispettive fasi operative sono differenti. Ne parleremo più dettagliatamente nel prossimo paragrafo.

In seguito all'esecuzione dello statement 40 un numero, che è il risultato del calcolo, sarà memorizzato nella locazione denominata Y1.

Passiamo ora all'esecuzione dello statement 50. Si osservi che in esso alla destra del segno uguale non compare soltanto la funzione; pertanto prima viene richiesta e calcolata la funzione, poi si somma ad essa 2*A. Nel nostro esempio, quando si calcola la funzione, le variabili fittizie vengono sostituite dai valori di B, C e D; quindi lo statement 50 dà come risultato lo stesso valore numerico dello statement

```
50   LET Y2 = (B↑2+C↑2+D↑2)↑2+2*A                    (7-4)
```

Allo stesso modo, lo statement 60 dà come risultato lo stesso valore numerico dello statement

```
60   LET Y3 = (Y1↑2+Y2↑2+Y2↑2)↑2                     (7-5)
```

Si ricordi che, prima che fosse preso in considerazione lo statement 60, i valori erano stati memorizzati nelle locazioni di memoria Y1 e Y2. Nel prossimo paragrafo daremo altri particolari sul calcolo delle funzioni.

7-1-1. Più funzioni in un programma

Un programma scritto in BASIC può contenere molte funzioni. Consideriamo ad esempio la sequenza di programma

```
10   DEF FNA(Z) = Z↑2+2*Z+1
20   DEF FNB(X,Y) = (X↑2+Y↑3)↑2
30   INPUT A,B,C
40   LET D = FNA(B)
50   LET G = FNB(C,D)
60   LET D1 = FNB(D,A)
70   LET D2 = FNB(D1,G)
```

(7-6)

Due o più funzioni diverse possono utilizzare le stesse variabili fittizie; comunque, quando viene richiesta una funzione, i valori numerici appropriati sono assegnati alle variabili fittizie, e si calcola la funzione. Quand'anche un'altra funzione usasse la

stessa variabile fittizia, non sarebbe un problema, perché le funzioni non interagiscono fra loro. Quindi, nel programma (7-6), non sorgerebbe nessun problema se lo statement 70 fosse scritto così:

```
10   DEF FNA(X) = X↑2+2*X+1
```

7-1-2. Funzioni con altre variabili oltre alle variabili fittizie

La funzione può contenere altre variabili, oltre alle variabili fittizie. Consideriamo ad esempio la seguente sequenza di programma:

```
10   DEF FNX(X,Y) = (X↑2+Y↑4)↑(1/3)+D
20   INPUT A,B,C
30   LET D = A*B+C
40   LET G = FNX(A,B)
```

(7-7)

La funzione utilizza non solo le variabili fittizie, ma anche la variabile D. Si tenga presente che le variabili fittizie sono quelle contenute nella lista che segue il nome della funzione, per cui qualsiasi altra variabile posta alla destra del segno uguale nello statement di definizione di funzione *non* è una variabile fittizia. Queste variabili devono comparire nel programma principale e devono essere già state definite prima che la funzione sia richiesta. Ad esempio, nel segmento di programma (7-7), nell'esecuzione dello statement 40 viene utilizzato il valore attuale di D calcolato nello statement 30. Pertanto lo statement 40 è sostanzialmente equivalente a

```
40   LET G = (A↑2+B↑4)↑(1/3)+D
```
(7-8)

Noi abbiamo preso in esame il caso di una funzione definita su una sola linea posta all'inizio del programma; in realtà queste funzioni possono comparire in qualunque punto di un programma scritto in BASIC, per quanto sia buona regola di programmazione metterle all'inizio, perché così il programmatore le fuò facilmente individuare.

Quando si richiede una funzione, lo statement di richiesta può contenere non solo variabili, ma anche costanti. Ad esempio, nel segmento di programma (7-7) potremmo correttamente scrivere in BASIC:

```
50   LET H = FNX(A,5)
```

Ora, quando lo statement 50 è eseguito, nella funzione, invece della variabile fittizia Y, ci sarà 5, e pertanto lo statement 50 è equivalente allo statement

```
50   LET H = (A↑2+5↑4)↑(1/3)+D
```

7-2. FUNZIONI MULTILINEA

Capita spesso di dover lavorare con funzioni che non si possono scrivere su una sola linea. In questi casi, è possibile utilizzare funzioni articolate su più linee, dette *funzioni multilinea*. I concetti di base restano sostanzialmente gli stessi, mentre la forma differisce di poco da quella delle funzioni su una sola linea. Spieghiamoci con un esempio:

```
100   DEF FNA(X,Y)
110   LET X1 = X↑2−Y↑2
120   IF X1 > = 0 THEN 140
130   LET X1 = −X1
140   LET FNA = X1↑.5
150   FNEND                                          (7-9)
```

La funzione multilinea ha inizio con uno statement di definizione, che differisce dallo statement di definizione su una sola linea per il fatto che è costituito *soltanto* dal nome della funzione e dalla lista delle variabili fittizie; vengono poi degli statements in BASIC che definiscono il calcolo della funzione. Si noti che *il valore della funzione dev'essere calcolato.* Nell'esempio, questo avviene nello statement 140, nel quale abbiamo

```
140   LET FNA = X1↑.5
```

Ovvero, il nome della funzione dev'essere posto uguale ad una quantità quale un'espressione o una costante; si noti che qui la lista delle variabili fittizie *non* è associata al nome della funzione. Infine dev'esserci uno statement di fine funzione, che è sempre lo stesso per tutte le funzioni multilinea, ed è formato dal numero di statement seguito dalla parola FNEND.

Una funzione multilinea può trovarsi in qualsiasi punto del programma, ma, come per le funzioni su una sola linea, è buona regola di programmazione collocare tutte le funzioni all'inizio (o alla fine), in modo che il programmatore le possa individuare facilmente. Vediamo con un esempio l'uso di una funzione multilinea.

```
 10  REM ESEMPIO DI PROGRAMMA CON
 15  REM UNA FUNZIONE MULTILINEA
 20  DEF FNA(X,Y)
 30  LET X1 = X↑2 - Y↑2
 40  IF X1 < = 0 THEN 60
 50  LET X1 = -X1
 60  LET FNA = X1↑.5
 70  FNEND
 80  REM INIZIO DEL PROGRAMMA
 85  REM PRINCIPALE
 90  INPUT A,B,C
100  LET G = FNA(A,B)+C↑2
110  PRINT G
120  END                                   (7-10)
```

All'esecuzione del programma, in un primo momento gli statements 20-70 vengono ignorati, dal momento che definiscono una funzione: soltanto quando viene eseguito lo statement 100 la funzione entra nell'esecuzione del programma.

La funzione multilinea, come già la funzione su una sola linea, può usare variabili desunte dal programma principale, che non compaiono nello statement di richiesta. Ad esempio, consideriamo quanto segue:

```
20  DEF FNB(X,Y)
30  LET FNB = X+2Y+A
40  LET FNB = FNB+X↑2
50  FNEND
60  INPUT A,B,C
70  LET Z = FNB(B,C)+2*B*C               (7-11)
```

Quando lo statement 70 è eseguito, la FNB sarà eseguita con i valori numerici di B e di C assegnati rispettivamente ad X ed Y e con il valore di A quale è quello memorizzato nella locazione di memoria relativa, dato che il valore di A usato nella funzione non è quello di una variabile fittizia. Si tenga presente che tutte le variabili usate da una funzione devono essere definite prima che la funzione sia richiesta.

Nell'esecuzione di una funzione multilinea può capitare che vengano cambiati i valori delle variabili del programma principale, se queste si trovano alla *sinistra* del segno uguale in uno statement di LET. Ad esempio, consideriamo quanto segue:

```
10  DEF FNA(X,Y)
20  LET X = X+2
30  LET B = B+4
40  LET FNA = X*B+Y
```

```
50  FNEND
60  LET A = 1
70  LET B = 2
80  LET C = 5
90  LET D = FNA(A,C)
```

(7-12)

Quando la funzione è eseguita i valori numerici di A e di C vengono assegnati alle variabili X ed Y della funzione. All'esecuzione dello statement 90 è richiesta la funzione, e lo statement 20 è numericamente equivalente allo statement

```
20  LET A = A+2
```

Può sembrare che il valore memorizzato di A possa venir modificato, ma non è così: il calcolatore, nel calcolare la funzione, lavora non con A, ma con X, al quale è stato assegnato il valore numerico di A. In altre parole, le variabili fittizie sono isolate dalle variabili del programma principale.

La situazione è completamente diversa quando si lavora con variabili desunte dal programma principale, che non sono variabili fittizie: ad esempio lo statement 30 cambia il valore di una variabile che non è una variabile fittizia. *Quest'operazione cambierà il valore memorizzato di quella variabile.* Ciò vuol dire che, prima che venga eseguito lo statement 90, il valore memorizzato di B sarà 2; dopo l'esecuzione dello statement 90, sarà 6: quindi, se in seguito B viene usata nel programma principale, il suo valore sarà 6. In certi casi questi cambiamenti sono utili, ma molto spesso generano confusione: se non volete cambiare i valori delle variabili del programma principale, non usatele alla sinistra del segno uguale nella funzione. Se ad esempio non si vuole che avvengano variazioni di questo tipo nel segmento (7-12), al posto delle linee 30 e 40 potremo scrivere:

```
30  LET B2 = B+4
40  LET FNA = X*B2+Y
```

(7-13)

Qui abbiamo definito una nuova variabile B2, e, in tal modo, non abbiamo cambiato il valore di B. Oppure potremmo usare tre variabili nella lista delle variabili fittizie, e servirci di questa lista per introdurre il valore numerico di B.

7-2-1. Uso degli statements di controllo

Con le funzioni multilinea si possono usare gli statements di controllo; ma il con-

trollo può essere trasferito solo da un punto all'altro della funzione, non ad uno statement posto al di fuori di essa.

7-3. LE FUNZIONI DI LIBRERIA NEL BASIC

Nel caso di calcoli che vadano eseguiti ripetutamente, il programmatore può scrivere una funzione per eliminare dal programma passaggi che si ripetono. Esistono delle funzioni che possono comparire in molti programmi. Per risparmiare ai programmatori tempo e fatica, il linguaggio BASIC fornisce dei programmi già pronti che il programmatore può richiedere senza dover scrivere una funzione. Queste funzioni si usano esattamente come quelle di cui si è parlato nei paragrafi 7-1 e 7-2, cioè richiedendole, ma il programmatore *non* ha bisogno di scriverle: queste funzioni sono dette *funzioni di libreria* del BASIC, o *funzioni intrinseche* del BASIC.

Vediamo come si usano. La funzione di libreria che dà il seno in un angolo in radianti è

```
SIN(X)                                                    (7-14)
```

Consideriamo ad esempio il seguente segmento di programma in BASIC:

```
10  LET Y = 1
20  LET A = SIN(Y)
```

(7-15)

All'esecuzione del programma A sarà uguale a sen(1 radiante) = sen(180°/π) = 0.841471.

Una funzione di libreria BASIC viene richiesta allo stesso modo di una funzione ordinaria, cioè essa appare unicamente alla destra del segno uguale in un'espressione. In generale, le funzioni di libreria BASIC hanno una sola variabile fittizia, e allora nello statement di richiesta deve comparire una sola variabile, che, naturalmente, va definita prima dell'esecuzione dello statement di richiesta. Si ricordi che, quando si usa una funzione di libreria, il programmatore deve soltanto richiederla, senza preoccuparsi di scriverla.

Vediamo qualche altro esempio, Come per le funzioni ordinarie, alla destra del segno uguale può comparire dell'altro oltre alla funzione. Ad esempio,

```
10  LET A =1
20  LET B = (SIN(A)+5)↑2                                  (7-16)
```

La variabile in una funzione di libreria può essere sostituita da un'espressione. Ad esempio, è corretto scrivere

```
10  LET A = 2.5
20  LET X = 5
30  LET B = 2A*SIN(A*X↑2)+3*X
```

Vediamo ora qualche altra funzione di libreria:

```
10  LET A = COS(X)                                    (7-17)
```

fa sì che A sia posta uguale al coseno di X in radianti. Analogamente,

```
10  LET B = TAN(X)                                    (7-18a)
```

e

```
20  LET B = COT(X)                                    (7-18b)
```

pongono B uguale rispettivamente alla tangente ed alla cotangente di x in radianti. Un'altra funzione di libreria BASIC è l'arcotangente:

```
20  LET A = ATN(X)                                    (7-19)
```

che pone A uguale all'arcotangente di X. Questa funzione BASIC è tale per cui, se X < 0, A sarà compresa fra $-$ /2 e 0 radianti; se invece X > 0, A sarà compresa fra 0 e $\pi/2$ radianti.

Altre funzioni di libreria BASIC concernono gli esponenziali ed i logaritmi. Ad esempio,

```
20  LET B = LGT(A)                                    (7-20)
```

pone B uguale a $\log_{10}A$. Si tenga presente che alcuni compilatori BASIC usano CLG(A) invece di LGT(A). Analogamente

```
20  LET C = LOG(A)                                    (7-21)
```

pone C uguale a $\log_e A = \ln A$ (il logaritmo naturale).

```
40  LET D = EXP(T)                                    (7-22)
```

pone D uguale ad e^T.

Molte volte si desidera sostituire ad un numero il suo valore assoluto. Il che vuol dire che, se il numero è positivo, si ottiene la sua grandezza; se è negativo, per ottenere il suo valore assoluto lo si moltiplica per −1: negli statements 130-150 del programma (5-25) si può vedere un segmento di programma che esegue questa operazione. C'è comunque una specifica funzione di libreria BASIC che realizza questa operazione: ABS(X). Consideriamo ad esempio

```
20   LET B = ABS(Y)                                    (7-23)
```

Se Y è positivo, come risultato B sarà posto uguale ad Y; se invece è negativo, B viene posto uguale a −Y.

La funzione SQR(X) calcola la radice quadrata di X. Voi forse chiederete perché darsi la pena di escogitare una funzione siffatta, dal momento che la radice quadrata si ricava facilmente in BASIC: ad esempio i due statements

```
20   LET B = X↑.5                                      (7-24a)
```

e

```
20   LET B = SQR(X)                                    (7-24b)
```

sono equivalenti. Ma, in linea generale, la forma (7-24b) è da preferirsi alla (7-24a) perché più efficiente, nel senso che il calcolo è eseguito più rapidamente. Si osservi che l'algoritmo usato nella (7-24a) è un algoritmo generale che serve per elevare un numero ad una potenza qualsiasi, mentre la (7-24b) si usa unicamente per ricavare la radice quadrata, e quindi per essa si può scrivere un algoritmo più specifico e di conseguenza più efficiente.

A volte è utile troncare un numero, cioè ometterne le cifre decimali: la funzione di libreria BASIC che esegue questa operazione è INT(X). Ad esempio,

```
20   LET A = INT(B)                                    (7-25)
```

dà questo risultato: se B = 21.632, A = 21. Alcune versioni di BASIC danno risultati diversi nel caso di numeri negativi: ad esempio, se A = −21.632, la funzione INT darà −21 nei casi in cui la parte decimale è semplicemente omessa; con altri compilatori invece la funzione darà −22, cioè INT(B) sarà il più grande numero intero minore o uguale a B. (Si tenga presente che −22 è minore di −21.) La differenza viene in luce solo con i numeri negativi, per cui è opportuno consultare il manuale BASIC del calcolatore utilizzato per sapere la forma assunta dalla funzione di libreria INT con i numeri negativi.

La funzione INT non si usa solo per troncare un numero, ma anche per arrotondarlo. Supponiamo che A sia un numero positivo, che noi vogliamo arrotondare: 21.6, ad esempio, si arrotonderà a 22, dato che, se la parte decimale è uguale o maggiore di 0.5, l'arrotondamento si fa al numero intero immediatamente successivo. Quindi, per arrotondare un numero positivo, non abbiamo che da sommare al numero 0.5, e poi troncare il numero ottenuto. Ad esempio,

```
20   LET A = INT(B+.5)
```
(7-26a)

dà il risultato desiderato. Se B è negativo, avremo

```
20   LET A = INT(B—0.5)
```
(7-26b)

Questo procedimento si basa sul presupposto che i numeri vengono arrotondati omettendo la parte decimale.

Un'altra utile funzione è SGN(X), che è definita nel modo seguente:

se $X > 0$ allora SGN(X) = 1
se $X < 0$ allora SGN(X) = -1
se $X = 0$ allora SGN(X) = 0 (7-27)

Per spiegare l'uso della funzione SGN, scriviamo un'espressione per arrotondare un numero sia positivo che negativo. Ecco un'espressione che esegue questa operazione:

```
20   LET A = SGN(B)*INT(ABS(B)+0.5)
```
(7-28)

Qui noi prendiamo il valore assoluto di B, poi lo arrotondiamo, quindi il numero risultante viene moltiplicato per SGN(B), allo scopo di ottenere il segno esatto. Si noti che questa espressione darà il risultato esatto qualunque sia la procedura di troncamento utilizzata dal compilatore.

Lo statement (7-28) mostra che una funzione di libreria BASIC può richiedere un'altra funzione di libreria. Ad esempio, quando è eseguita la parte dello statement (7-28) posta alla destra del *, dapprima viene richiesta la funzione ABS, ottenendo un risultato, poi si somma 0.5 a questo risultato, ed infine viene richiesta ed eseguita la funzione INT.

Spesso occorre arrotondare un determinato numero di cifre decimali, non volendo cioè eliminarle tutte. Supponiamo ad esempio di voler eseguire un arrotondamento fino a tre cifre decimali. Potremo farlo con il seguente statement:

```
20   LET A = SGN(B)*INT(1000*ABS(B)+0.5)/1000
```
(7-29)

Esaminiamolo. Supponiamo che B = −26.36759. Moltiplicando il suo valore assoluto per 1000, si ottiene 26367.59. Poi a quest'ultimo numero si aggiunge 0.5, ottenendo 26368.09, che, eseguendo la funzione INT, è troncato a 26368. Dividendo questo numero per 1000, si ottiene 26.368, moltiplicando il quale per SGN(B) si ottiene −26.368, cioè il grado di arrotondamento desiderato. In generale, per arrotondare un numero fino a k cifre decimali, dobbiamo sostituire 1000 (10^3) con 10k nei due posti in cui 1000 compare nello statement (7-29).

In determinate applicazioni interessa ricavare un numero in modo casuale. Supponiamo ad esempio di voler simulare il lancio di un dado; ci proponiamo quindi di ottenere dei numeri dall'1 al 6 in modo casuale. Abbiamo una funzione di libreria BASIC che ci permette di ottenere numeri casuali: la funzione RND. Prendiamo ad esempio

```
20   LET B = RND                                        (7-30)
```

B sarà un numero casuale. La funzione di libreria RND dà un numero *decimale* compreso fra 0 e 1. È possibile generare una sequenza di questi numeri. Ad esempio, con

```
10   DIM B(100)
20   FOR I = 1 TO 100
30   LET X(I) = RND
40   NEXT I                                             (7-31)
```

il vettore X(I) conterrà alla fine una lista di 100 numeri casuali, tutti compresi fra 0 e 1.

Supponiamo di voler simulare il lancio di un dado: i numeri casuali dovranno essere numeri interi compresi fra 1 e 6. Questo si può ottenere con il seguente statement:

```
LET A = INT(1+6*RND)                                    (7-32)
```

RND sarà un numero compreso fra 0.00000001 e 0.999999999. Quindi 1+6 RND sarà compreso fra 1.000000006 e 6.999999994. Applicando la funzione INT, A sarà un numero intero compreso fra 1 e 6.

Si noti che la funzione RND si usa spesso per generare dati casuali, con cui poi verificare i programmi. Quando si verifica un programma, conviene non usare dati particolari, dal momento che il programma potrebbe lavorare, per un caso fortuito, con questi stessi dati.

I numeri forniti dalla funzione RND non sono veramente casuali. Ad esempio, se esaminiamo tutti i valori di X generati dal segmento di programma (7-31), troviamo che i numeri sono casuali; tuttavia, ad ogni esecuzione del segmento di programma

(7-31), risulterà la *medesima* sequenza "casuale". In alcune applicazioni questo è utile; non lo è, se si vuol veramente simulare un processo casuale. Si può variare la sequenza generata dalla funzione di libreria RND mediante lo statement RANDOMIZE. Vediamo ad esempio il seguente segmento di programma:

```
 5  DIM X(100),Y(100)
10  RANDOMIZE
20  FOR I = 1 TO 100
30  LET X(I) = RND
40  NEXT I
50  RANDOMIZE
60  FOR I = 1 TO 100
70  LET Y(I) = RND
80  NEXT I
```

(7-33)

Ora la sequenza sarà diversa per X e per Y; inoltre, ad ogni esecuzione del programma, sarà generata per X ed Y una sequenza differente.

In questo paragrafo abbiamo illustrato le più comuni funzioni di libreria BASIC. Controllate sul manuale di BASIC annesso al calcolatore utilizzato se ve ne sono delle altre, specifiche per quel calcolatore. Si tenga presente che su qualche calcolatore alcune funzioni di libreria BASIC non sono utilizzabili. Anche su questo punto occorre consultare il manuale.

7-4. LE SUBROUTINES IN BASIC

Quando le stesse operazioni si ripetono più volte in un programma, si può evitare di riscrivere gli statements mediante una *sottoprocedura* BASIC (*subroutine* in inglese). La subroutine si distingue dalla funzione per la modalità di richiesta e per il modo in cui opera. Conviene usare le subroutines soprattutto quando si ha a che fare con vettori. Esamineremo ora alcuni statements BASIC che si usano con le subroutines, per analizzarne poi le modalità operative.

Non c'è uno statement che indica l'inizio di una subroutine; ce n'è invece uno che ne segnala la fine: lo statement di RETURN. La sua forma è

```
600  RETURN
```

È formato cioè dal numero di statement seguito dalla parola-chiave RETURN. Una subroutine è richiesta mediante uno statement che ha questa struttura:

```
30   GOSUB 150                                          (7-34)
```

La sua forma è: numero di statement, la parola-chiave GOSUB, seguita da un numero. Questo numero deve corrispondere al numero di uno statement del programma BASIC, che sarà di fatto il primo statement della subroutine.

L'esempio seguente illustra l'uso delle subroutines. Scriviamo una subroutine che divide ciascun elemento di un vettore per il suo valore medio: il nostro scopo è di stampare una tabella in cui siano elencati, l'uno accanto all'altro, il vettore originario ed il vettore così calcolato, e di ripetere almeno due volte questo calcolo. Introduciamo allora i dati per due vettori; *poi* sarà stampata l'informazione richiesta. (Si tenga presente che i due vettori introdotti non interagiscono fra loro.)

```
 10  REM PROGRAMMA CON UNA SUBROUTINE
 20  DIM A(100),B(100),S(100)
 30  PRINT "INTRODURRE LA DIMENSIONE",
 35  PRINT "DEL VETTORE A"
 40  INPUT N
 50  PRINT "INTRODURRE IL VETTORE A"
 60  FOR I = 1 TO N
 70  INPUT A(I)
 80  NEXT I
 90  PRINT "INTRODURRE LA DIMENSIONE",
 95  PRINT "DEL VETTORE B"
100  INPUT M
110  PRINT "INTRODURRE IL VETTORE B"
120  FOR I = 1 TO M
130  INPUT B(I)
140  NEXT I
150  REM ASSEGNA I VALORI
155  REM DELLA SUBROUTINE
160  LET T = N
170  FOR I = 1 TO N
180  LET S(I) = A(I)
190  NEXT I
200  REM RICHIAMA LA SUBROUTINE
210  GOSUB 500
220  REM STAMPA IL CONTENUTO
225  REM DEL VETTORE A
230  PRINT "I", "A", "MED"
240  FOR I = 1 TO N
```

```
250  PRINT I,A(I),S(I)
260  NEXT I
270  REM ASSEGNA I VALORI
275  REM DELLA SUBROUTINE
280  LET T = M
290  FOR I = 1 TO M
300  LET S(I) = B(I)
310  NEXT I
320  REM RICHIAMA LA SUBROUTINE
330  GOSUB 500
340  REM STAMPA IL CONTENUTO
345  REM DEL VETTORE B
350  PRINT "I","B","MED"
360  FOR I = 1 TO M
370  PRINT I,B(I),S(I)
380  NEXT I
390  STOP
500  REM INIZIO SUBROUTINE
510  LET Y = 0
520  GOR I = 1 TO T
530  LET Y = Y+S(I)
540  NEXT I
550  LET Y = Y/T
560  FOR I = 1 TO T
570  LET S(I) = S(I)/Y
580  NEXT I
590  RETURN
999  END                                    (7-35)
```

Esaminiamo le fasi operative del programma, il cui diagramma di flusso è rappresentato in Figura 7-1, analizzando dapprima la subroutine che comincia allo statement 500. In essa si sommano tutti gli elementi del vettore S: la somma è chiamata Y. Poi, nello statement 550, Y è diviso per T (il numero totale degli elementi), per cui Y viene ad essere la media di tutti gli elementi del vettore in questione. Nelle linee 560-580 ciascun elemento del vettore S è diviso per Y, per cui, al termine del calcolo, ciascun elemento del vettore S risulta sostituito dal suo valore originario diviso per il valore medio del vettore stesso.

Esaminiamo ora il programma. I vettori sono dimensionati nello statement 20. Negli statements 40-80 vengono introdotti N (dimensione del vettore A) e gli elementi che lo compongono; analogamente negli statements 100-140 sono introdotti M (dimensione del vettore B) e gli elementi che lo compongono.

Vogliamo ora che la subroutine "operi" sul vettore A. Richiedere la subroutine
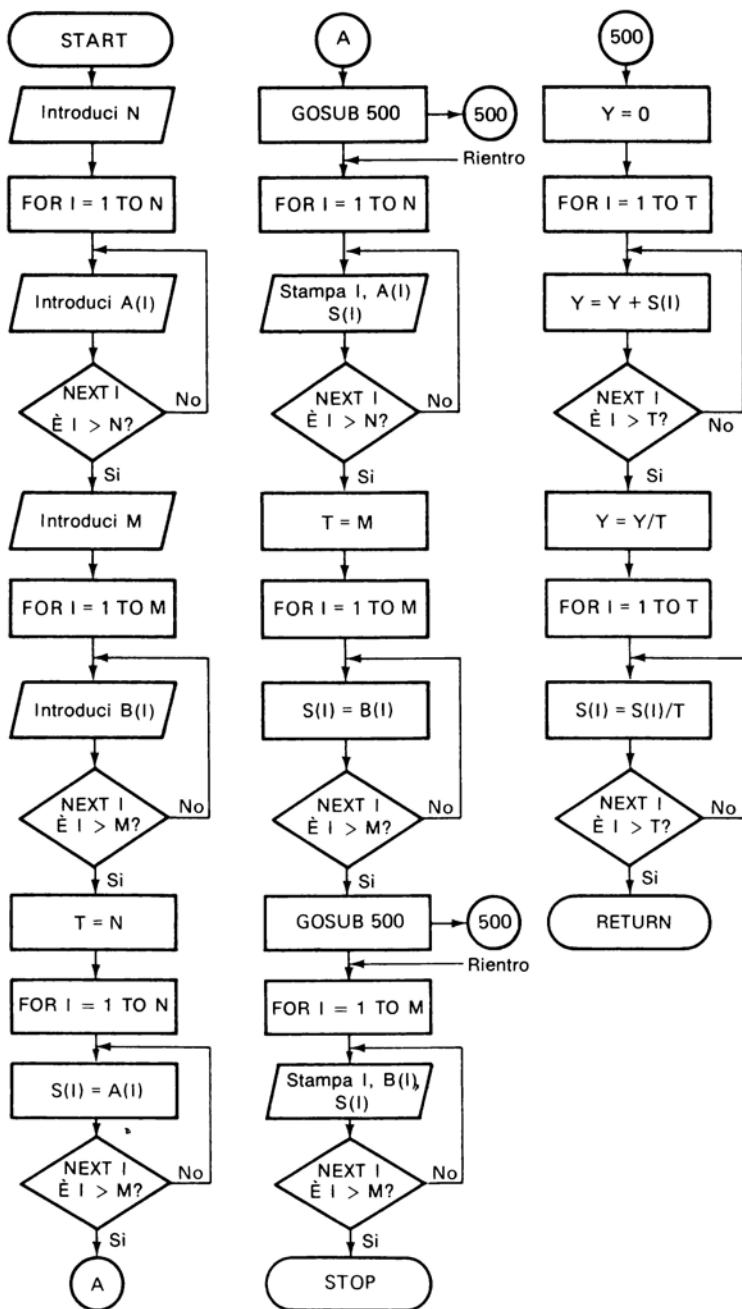
Figura 7.1 — Diagramma di flusso del programma (7-35) che illustra una subroutine.

con lo statement GOSUB 500 è quasi la stessa cosa che ricorrere ad uno statement GO TO 500, in quanto il controllo passerà allo statement 500: la differenza sostanziale è che, quando si usa lo statement di GOSUB, il calcolatore "se ne ricorda", per cui, quando si trova lo statement di RETURN, il controllo torna allo statement che segue lo statement di GOSUB. Quindi gli statements di una subroutine si comportano come se facessero parte del programma principale. Non vi sono, in altre parole, variabili fittizie; il che vuol dire che dovremo assegnare i valori appropriati alle variabili usate nella subroutine. Nel nostro esempio, il vettore S è proprio un vettore costituito da T elementi. Quindi, nella linea 160, assegneremo a T il valore memorizzato nella locazione di memoria N e, analogamente, con gli statements 170-190 assegneremo ad S i valori appropriati. A questo punto la subroutine è richiesta ed eseguita: in seguito all'esecuzione, i valori memorizzati nel vettore S saranno quelli del vettore A divisi per il relativo valore medio. Ora il controllo torna allo statement 220, che vien dopo lo statement che ha richiesto la subroutine, mentre negli statements 230-260 vengono stampate le quantità I, A(I) ed S(I). Si osservi che S(I) può essere trattata come una variabile ordinaria.

Vogliamo ora ripetere i calcoli con il vettore B. Negli statements 280-310 vengono assegnati i valori appropriati a T ed al vettore S; poi viene di nuovo richiesta ed eseguita la subroutine. Si osservi che il controllo torna sempre allo statement che vien dopo lo statement di richiesta. Infine gli statements 350-380 faranno stampare I, B(I) ed S(I). Si osservi che il vettore S(I) conterrà ora gli elementi del vettore B divisi per il relativo valore medio.

Supponiamo di voler usare i valori del vettore A divisi per il relativo valore medio in un momento successivo del programma, e inoltre di richiamare la subroutine, prima di questo momento: occorre allora assegnare i valori memorizzati nel vettore S ad un altro vettore. Ad esempio, potremmo aggiungere gli statements

```
262  FOR I = 1 TO N
264  LET D(I) = S(I)
266  NEXT I                                    (7-36)
```

Analoghi statements sono necessari per memorizzare gli elementi del vettore B divisi per il suo valore medio. Va da sé che dovremmo dimensionare il vettore D.

7-4-1. Uso degli statements di controllo

Con le subroutines si possono usare statements di controllo come GO TO e IF-THEN, esattamente come con tutti i programmi in BASIC. Ma ci sono due regole da osservare:

1) Il controllo *non può* passare da uno statement *interno* alla subroutine ad uno *esterno* ad essa (tranne che con lo statement di RETURN);

2) Il controllo *non può* passare da uno statement del programma principale ad uno della subroutine (tranne che con lo statement di GOSUB).

7-4-2. Uso dello statement di STOP

Consideriamo lo statement di STOP del programma (7-35) nella linea 390. Se non ci fosse, l'esecuzione continuerebbe con la subroutine, il che, naturalmente, sarebbe errato. Ad esempio, si troverebbe uno statement di RETURN, che non potrebbe essere eseguito, non essendoci stato uno statement di GOSUB a richiedere la subroutine. Quindi, quando si usano subroutines, bisogna sempre inserire opportunamente uno o più statements di STOP. Naturalmente l'ultimo statement del programma dev'essere uno statement di END.

7-4-3. Più subroutines in un programma

In un programma può comparire più di una subroutine, ciascuna delle quali deve terminare con uno statement di RETURN. Il numero di statement nello statement di GOSUB indica qual è la subroutine richiesta. Appena s'incontra uno statement di RETURN, il controllo torna allo statement successivo allo statement di GOSUB che ha trasferito il controllo alla subroutine.

7-5. ANNIDAMENTO DI SUBROUTINES

Una subroutine può richiedere un'altra subroutine: la seconda subroutine è detta subroutine *annidata*. Valgono le stesse regole delle subroutines singole, cioè uno statement di GOSUB trasferisce il controllo allo statement che porta il numero indicato, e, quando si trova uno statement di RETURN, si ritorna allo statement successivo allo statement di GOSUB che ha richiesto la subroutine la cui esecuzione è appena terminata. Il seguente segmento di programma servirà ad illustrare questi concetti:

```
        .
30  LET A = X
40  LET D = X+4
50  GOSUB 500
```

```
 60  LET C = B
      .
100  LET A = 20
110  LET D = 2*A+4
120  GOSUB 400
130  LET F = B
      .
180  LET T = W
190  GOSUB 500
      .
250  STOP
400  REM PRIMA SUBROUTINE
410  LET T = A↑2+D↑2
420  GOSUB 500
430  LET B = T↑2+1
440  RETURN
500  REM SUBROUTINE ANNIDATA
510  IF T > 50 THEN 530
520  T = T+20
530  T = T↑2
540  RETURN
999  END                                    (7-37)
```

Esaminiamo le fasi operative di questo segmento di programma. Negli statements 30 e 40 sono assegnati ad A e a D dei valori che saranno usati nella subroutine, supponendo che ad X sia stato assegnato un valore in precedenza. Poi lo statement di GOSUB alla linea 50 passa il controllo allo statement 400. Nello statement 410 viene calcolato T, e subito dopo, con lo statement di GOSUB della linea 420 il controllo passa allo statement 500, che è la subroutine annidata. In questa seconda subroutine i calcoli sono eseguiti su T e, quando s'incontra lo statement di RETURN (linea 540), il controllo ritorna allo statement 430: si ricordi che lo statement 420 ha richiesto la subroutine annidata, per cui, quando s'incontra lo statement di RETURN, il controllo torna allo statement successivo allo statement di GOSUB che ha richiesto la subroutine la cui esecuzione è appena terminata. A questo punto viene eseguito lo statement 430, e subito dopo s'incontra lo statement 440, che passa il controllo allo statement 60. Il controllo a questo punto torna di nuovo allo statement successi-

vo allo statement di GOSUB che ha richiesto la subroutine la cui esecuzione è appena terminata.

All'esecuzione dello statement 120, il ciclo precedentemente descritto si ripete: è eseguito lo statement 410, poi il controllo passa allo statement 500 ed è eseguita la seconda subroutine; quindi il controllo ritorna allo statement 430, ed al termine dell'esecuzione della subroutine il controllo passa allo statement 130.

Si noti che nello statement 190 GOSUB 500 la seconda subroutine è richiesta direttamente dal programma principale: in casi come questo la seconda subroutine è eseguita indipendentemente dalla prima.

Quando due subroutines sono inserite l'una nell'altra, la prima richiede la seconda, la quale *non può* richiedere la prima, cioè non può esserci una procedura ciclica di controllo fra gli statements di GOSUB e di RETURN.

7-6. PIÙ RETURN DA UNA SUBROUTINE

Una subroutine può usare statements di controllo, quindi può avere diverse diramazioni, ciascuna terminante con uno statement di RETURN; non appena è eseguito uno di questi, il controllo lascia la subroutine e passa allo statement che vien dopo lo statement di GOSUB. Vediamo un esempio:

```
 10  REM ILLUSTRAZIONE DELLA SUBROUTINE
 15  REM A PIU' RETURN
 20  INPUT A,B,C,D
 30  LET A1 = A
 40  LET B1 = B
 50  GOSUB 200
 60  LET F = X
 70  LET A1 = C
 80  LET B1 = D
 90  GOSUB 200
100  LET G = F+X
110  PRINT G
120  STOP
200  REM SUBROUTINE A PIU' RETURN
210  LET X = A1↑2-B1↑2
220  IF X > 0 THEN 250
230  LET X = X+50
240  RETURN
250  LET X = X-10
260  RETURN
999  END                                        (7-38)
```

Esaminiamo il programma. Vengono introdotte quattro variabili; poi, negli statements 30 e 40, vengono assegnati dei valori alle variabili A1 e B1, che vengono usate nella subroutine. Lo statement 50 passa il controllo allo statement 200, che è il primo statement della subroutine. Nello statement 210 viene calcolato X. Se X $\leqslant$ 0, X viene incrementato di 50 nello statement 230 (LET X = X+50), e successivamente è eseguito lo statement di RETURN alla linea 240, con cui il controllo torna allo statement 60. Se invece il valore di X calcolato nello statement 210 è positivo, il controllo va allo statement 250, e subito dopo è eseguito lo statement di RETURN alla linea 260, con cui il controllo torna allo statement 60. Concludendo, *ogni* statement di RETURN di una subroutine a molti rami opera allo stesso modo dello statement di RETURN di una subroutine con un solo RETURN.

ESERCIZI

7-1. Scrivete una funzione su una sola linea che calcoli la media di quattro variabili. Verificate la funzione scrivendo un opportuno programma in BASIC ed eseguendolo su di un calcolatore.

7-2. Come vengono assegnate le variabili fittizie di una funzione?

7-3. Si consideri lo statement

```
10   DEF FNA(X,Y,Z) = X↑2+Y↑2+Y*Z+A
```

Come viene assegnato il valore di A?

7-4. Lo statement dell'Esercizio 7-3 può essere eseguito prima che sia stato assegnato il valore di A?

7-5. Lo statement dell'Esercizio 7-3 può essere richiesto prima che sia stato assegnato il valore di A?

7-6. Perché la sequenza seguente è inesatta?

```
10   DEF FNB(A,B) = (A*B+Z)↑3
20   LET X = 2
30   LET C = FNB(X,Y)+2*X↑2
40   LET Y = 3
```

7-7. Scrivete una funzione che calcoli e stampi le radici di un'equazione di secondo grado. (N.B.: V. paragrafo 4-4.) Verificate i risultati ottenuti scrivendo un programma opportuno ed eseguendolo su di un calcolatore.

7-8. Scrivete una funzione che dia come risultato il valore più alto fra quattro elementi introdotti da tastiera. Verificate i risultati ottenuti scrivendo un opportuno programma in BASIC ed eseguendolo su di un calcolatore.

7-9. Scrivete uno statement BASIC che esegua

$$y = 2 \text{ sen } x + 4 \text{ tg } y$$

in cui x ed y sono introdotte in gradi (1 grado = 0.01745329 radianti).

7-10. Scrivete un programma in BASIC che calcoli

$$y = (2a^2 + 3b^2 - 7c^2)^{1/3}$$

e stampi il valore di y con due cifre decimali. I valori di a, b e c devono essere introdotti da telescrivente. Verificate il programma eseguendolo su di un calcolatore.

7-11. Scrivete uno statement che calcoli

$$y = \text{sen}(a+be^c)$$

7-12. Scrivete una subroutine che determini il massimo valore presente in un vettore a due dimensioni. La dimensione del vettore dev'essere introdotta come dato, ma non dev'essere maggiore di 100 × 100. Verificate i risultati ottenuti scrivendo un opportuno programma in BASIC ed eseguendolo su di un calcolatore.

7-13. Ripetete l'Esercizio 6-8, ma utilizzando questa volta una subroutine. Verificate i risultati ottenuti scrivendo un opportuno programma in BASIC ed eseguendolo su di un calcolatore.

7-14. Ripetete l'Esercizio 6-9, ma utilizzando una subroutine. Verificate i risultati ottenuti scrivendo un opportuno programma in BASIC ed eseguendolo su di un calcolatore.

7-15. Ripetete l'Esercizio 6-16, ma utilizzando una subroutine. Verificate i risultati ottenuti scrivendo un opportuno programma in BASIC ed eseguendolo su di un calcolatore.

7-16. Ripetete l'Esercizio 6-17, ma utilizzando una subroutine. Verificate i risultati ottenuti scrivendo un opportuno programma in BASIC ed eseguendolo su di un calcolatore.

CAPITOLO 8

TRATTAMENTO DEI CARATTERI ALFANUMERICI

Il linguaggio BASIC esegue fondamentalmente operazioni matematiche; tuttavia può lavorare anche su informazioni alfanumeriche, cioè può trattare stringhe, ossia sequenze di lettere e numeri. Ad esempio possiamo scrivere un programma in BASIC che metta in ordine alfabetico una lista di nomi. In questo capitolo vedremo alcuni esempi di trattamento di dati alfanumerici, ma con l'avvertenza che occorre star bene attenti perché non tutte le operazioni su informazioni alfanumeriche sono lecite su tutti i calcolatori.

8-1. VARIABILI ALFANUMERICHE. STRINGHE

Sequenze di lettere, di numeri ed, eventualmente, di simboli, sono dette stringhe alfanumeriche, o semplicemente stringhe. Ad esempio,

"CARLO ROSSI"
"CAMERA 229"
"CARLO, LUIGI, ALDO" (8-1)

sono stringhe. Quando delle stringhe sono usate negli statement di DATA o in risposta ad uno statement di INPUT, alcune versioni di BASIC permettono di omettere le virgolette che racchiudono le stringhe stesse. Se i dati alfanumerici comprendono virgole o altri simboli del BASIC, questi vanno sempre posti fra virgolette.

Per rappresentare le stringhe alfanumeriche possiamo servirci di variabili, che dovranno avere una forma particolare, cioè una lettera seguita dal simbolo del dollaro $. Ad esempio,

```
A$
B$
X$                                                    (8-2)
```

sono variabili-stringa BASIC corrette. Alcuni compilatori BASIC ammettono una lettera ed un numero intero prima del simbolo del dollaro: con questi compilatori variabili-stringa come D3$, A2$, X3$, etc. saranno valide.

Il numero di caratteri che può venir memorizzato in una variabile-stringa varia da un calcolatore all'altro; la maggior parte permette un massimo di 15 caratteri, ma ce ne sono altri che ammettono fino a 4095 caratteri. Si tenga presente che gli spazi bianchi sono considerati caratteri; quindi

"CARLO ROSSI"

è formata da undici caratteri.

L'ingresso e l'uscita dei dati relativamente ai caratteri alfanumerici avvengono sostanzialmente nello stesso modo che con i numeri. Vediamo ad esempio il programma seguente:

```
10  READ A$,B$,C$
20  PRINT A$,C$,B$
30  DATA "LIBRO","CASA","TAVOLO"
40  END                                               (8-3)
```

Quando si esegue il programma, all'esecuzione dello statement di PRINT si avrà come risultato:

```
LIBRO       TAVOLO       CASA                         (8-4)
```

Si noti che la spaziatura ed il significato delle virgole nello statement di PRINT sono sostanzialmente quelli che abbiamo illustrato nel paragrafo 3-4; così, per avere una spaziatura più stretta, si userà il punto e virgola, etc.

Un altro esempio:

```
10  INPUT A$,B,C,D
20  LET E = (B+C+D)/3
30  PRINT A$,"MEDIA =",E
40  END                                               (8-5)
```

Vediamo l'esecuzione di questo programma. Sulla telescrivente apparirà quanto segue:

```
>RUN
MED         12:30       4-MAR-77
? ROSSI,100,90,80
ROSSI       MEDIA =     90              (8-6)
```

L'informazione battuta da chi utilizza il programma è costituita da RUN, e dai dati ROSSI,100,90,80 dopo il punto interrogativo. Si tenga presente che le stringhe alfanumeriche possono essere associate a numeri e che il testo può essere stampato nel modo solito.

8-1-1. I vettori

Con le stringhe alfanumeriche si possono usare i vettori, essendo il nome del vettore formato da una sola lettera seguita da $. Vediamo ad esempio il seguente semplice programma:

```
 10   DIM A$(20),B$(20)
 20   INPUT N
 30   FOR I = 1 TO N
 40   INPUT A$(I)
 50   NEXT I
 60   FOR J = 1 TO N
 70   LET B$(I) = A$(I)
 80   PRINT B$(I)
 90   NEXT I
999   END                                        (8-7)
```

In questo programma dimensioniamo il vettore A$ a 20 elementi; si tenga presente che, esattamente come per i vettori numerici, in assenza dello statement di dimensione il vettore A$ avrebbe 10 (o 11) elementi. Nel nostro programma introduciamo N stringhe (statements 30-50). Esaminiamo ora il ciclo che va dallo statement 60 al 90. Lo statement 70

```
70   LET B$(I) = A$(I)                           (8-8)
```

mostra che fra le variabili-stringa vale l'operatore di uguaglianza esattamente come fra le variabili ordinarie. Successivamente stampiamo B$(I), e quindi l'uscita sarà uguale all'ingresso.

Si tenga presente che la maggior parte delle versioni di BASIC ammette soltanto l'uso di vettori ad una dimensione.

È possibile porre una variabile-stringa uguale ad una "costante". Ad esempio,

```
75   LET D$ = "BIANCHI"                          (8-9)
```

La maggior parte dei compilatori vuole l'uso delle virgolette, in questo tipo di espressione, anche se si possono omettere quando i dati vengono introdotti.

Anche se fin qui abbiamo illustrato il caso di vettori ad una dimensione, è opportuno tener presente che con le variabili-stringa si possono usare anche vettori a due dimensioni.

8-1-2. Lo statement di RESTORE

Abbiamo parlato dello statement di RESTORE nel paragrafo 3-2. Si ricorderà che questo statement riporta il puntatore dei dati all'inizio del blocco di dati. In realtà, ci sono *due* puntatori dei dati, uno per le costanti numeriche e l'altro per le stringhe: ogni volta che viene letta una voce dei dati, il puntatore appropriato va alla voce seguente dello *stesso* genere. Quando viene eseguito lo statement di RESTORE, *entrambi* i puntatori sono riposizionati alla prima voce dei loro rispettivi blocchi di dati.

Volendo riposizionare un solo puntatore, dovremo modificare lo statement di RESTORE: se la parola RESTORE è seguita da $, sarà riposizionato il solo puntatore di stringa, mentre, se è seguita da un *, sarà riposizionato il solo puntatore dei dati numerici. Vediamo l'esempio seguente:

```
10   READ A$,B,C$,D
20   RESTORE
30   READ E$,F
40   RESTORE$
50   READ G$,H
60   RESTORE*
70   READ I$,J

      .
200  DATA ABC,23,DEF,99                    (8-10)
```

All'esecuzione dello statement 10, verranno effettuate le seguenti assegnazioni:

A$ = "ABC"
B = 23
C$ = "DEF"
D = 99

Lo statement 20 riposiziona entrambi i puntatori, per cui avremo queste assegnazioni:

E$ = "ABC"
F = 23

Ora il puntatore di numero individua 99, e il puntatore di stringa DEF: saranno queste le voci che verranno lette successivamente. Ma lo statement 40 riposiziona il puntatore di stringa in modo che indichi ora ABC. Quindi, all'esecuzione dello statement 50, avremo le assegnazioni seguenti:

```
G$ = "ABC"
H = 99
```

A questo punto lo statement 60 riposiziona il puntatore dei dati numerici, che così indicherà 23, mentre il puntatore di stringa, non essendo riposizionato, individuerà DEF. Pertanto, all'esecuzione dello statement 70, avremo:

```
I$ = "DEF"
J = 23
```

In questo paragrafo abbiamo presentato alcuni programmi molto semplici per illustrare alcuni dei concetti fondamentali sul trattamento delle stringhe alfanumeriche. Nel prossimo paragrafo vedremo programmi più complessi.

8-2. USO DEGLI STATEMENTS DI CONTROLLO CON LE VARIABILI—STRINGA

In questo paragrafo parleremo dell'uso degli statements di controllo con le variabili-stringa. Lo statement di GO TO non è condizionato dall'uso di questa o quella variabile, e quindi le sue modalità operative sono le stesse che con le variabili numeriche. Pertanto fermeremo la nostra attenzione sullo statement di IF-THEN.

Alcuni calcolatori sono in grado di utilizzare tutti gli operatori relazionali della Tabella 4-1 con le variabili-stringa, altri no: come sempre occorre riferirsi al manuale annesso al calcolatore. Comunque, in linea generale, tutti i compilatori BASIC ammettono l'uso dell'uguale (=) e diverso (< >) con le variabili-stringa. Vediamo ad esempio

```
40   IF A$ = B$ THEN 140
50   LET A$ = "ABC"                                   (8-11)
```

Se la variabile-stringa A$ è identica a B$, spazi bianchi compresi, verrà eseguito per primo lo statement 140; se A$ e B$ non sono identiche, il primo ad essere eseguito sarà lo statement 50.

Un'altra forma che si può usare è

```
 90   IF C$ < > "LIBRO QUI" THEN 300
100   -------                                         (8-12)
```

Ora, se C$ equivale esattamente a LIBRO QUI, compreso lo spazio bianco, per primo sarà eseguito lo statement 100. Se invece C$ non corrisponde esattamente a LIBRO QUI, sarà eseguito per primo lo statement 300.

Con le variabili-stringa non si possono usare le relazioni aritmetiche come addizione, etc.; ciò vuol dire che non possono comparire negli statements di IF-THEN alla destra del segno uguale relazioni di questo tipo.

Vediamo ora questi concetti applicati ad un programma. Supponiamo che vengano introdotti i nomi degli studenti che compongono una classe ed i voti da essi riportati in tre esercitazioni. Viene poi calcolata la media di ciascuno studente. Supponiamo ora non di voler stampare l'elenco dell'intera scolaresca, ma di voler introdurre il nome di un determinato studente, e di ottenere la stampa del suo nome e della sua media. Ed ecco il programa che esegue queste operazioni:

```
 10  REM PROGRAMMA CHE LOCALIZZA UN NOME
 20  DIM A$(100),B(100)
 30  PRINT "INTRODURRE IL NUMERO TOTALE",
 35  PRINT "DEGLI STUDENTI"
 40  INPUT N
 50  PRINT "INTRODURRE I NOMI ED I VOTI",
 55  PRINT "DEGLI STUDENTI"
 60  FOR I = 1 TO N
 70  INPUT A$(I),G1,G2,G3
 80  B(I) = (G1+G2+G3)/3
 90  NEXT I
100  REM INTRODUZIONE DEL NOME DELLO STUDENTE
105  REM E STAMPA DELLA MEDIA
110  PRINT "INTRODURRE IL NOME DELLO STUDENTE",
115  PRINT "DI CUI SI VUOL SAPERE LA MEDIA"
120  INPUT C$
130  FOR J = 1 TO N
140  LET J1 = J
150  IF A$(J) = C$ THEN 170
160  NEXT J
170  PRINT A$(J1);"MEDIA =";B(J1)
999  END                                      (8-13)
```

In Figura 8-1 è rappresentato il diagramma di flusso. Vediamo le fasi operative del programma. I vettori sono dimensionati nello statement 20; N, il numero degli studenti, è introdotto all'esecuzione dello statement 40. L'introduzione dei dati ed il calcolo delle medie avvengono nel ciclo definito dagli statements 60-90. In seguito all'esecuzione del ciclo, i nomi degli studenti vengono memorizzati nel vettore A$, le medie nel vettore B.
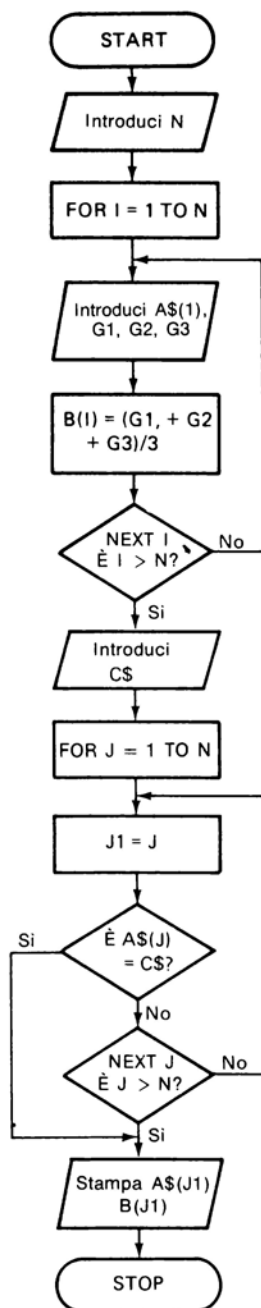
Figura 8.1 — Diagramma di flusso del programma (8-13).

All'esecuzione dello statement 120, il nome dello studente del quale vogliamo conoscere la media viene introdotto e memorizzato nella locazione di memoria denominata C$. Esaminiamo ora il ciclo degli statements 130-160. Nello statement 140 poniamo J1 = J: questa è la prima operazione che viene eseguita ad ogni giro del ciclo. Poi A$(J) viene confrontata con il nome dello studente che è stato introdotto come dato: se non sono uguali, l'esecuzione del ciclo continua e, quando si giunge ad avere l'appropriata A$(J), che corrisponde al nome introdotto, il controllo passa allo statement 170. La procedura ciclica ha termine e viene stampata l'informazione desiderata, cioè vengono stampate A$(J1) e B(J1). Si noti che si potrebbe aggiungere lo statement

```
180  GO TO 110
```

In tal modo si potrebbero introdurre i nomi di altri studenti, ed avere la stampa delle loro medie: il programma in questo caso terminerebbe nel modo che si è detto nel paragrafo 4-1.

Abbiamo visto l'uso degli operatori di relazione = e < >. In molti sistemi BASIC si possono usare anche <= e >=: le stringhe memorizzate cioè sono considerate come se fossero costituite da valori numerici, che si possono confrontare. Vediamo meglio questo punto. Le lettere sono in realtà memorizzate sotto forma di numeri, perché i numeri (binari) sono l'unico tipo d'informazione su cui il calcolatore lavora, Quando una variabile termina con $, vuol dire che il relativo numero memorizzato dovrà essere decodificato opportunamente sotto forma di lettera; allo stesso modo, introducendo un'informazione costituita da una lettera, questa viene opportunamente tradotta in un numero.

In BASIC le lettere vengono codificate in modo tale che A ha un valore numerico inferiore a quello di B, che ha a sua volta un valore numerico inferiore a quello di C, e così via; cioè, vengono numerate secondo l'ordine alfabetico. Pertanto AABC avrà un numero di codice minore di AADC, e così via: è con i numeri corrispondenti ai codici che si possono usare i segni di "maggiore di" e "minore di". Per spiegarne l'uso scriviamo un programma in BASIC che mette in ordine alfabetico i nomi di una lista.

```
10  REM PROGRAMMA CHE METTE IN
15  REM ORDINE ALFABETICO UNA LISTA
20  DIM A$(100)
30  PRINT "INTRODURRE IL NUMERO TOTALE"
40  INPUT N
50  PRINT "INTRODURRE I NOMI"
60  FOR I = 1 TO N
70  INPUT A$(I)
80  NEXT I
90  FOR I = 1 TO N
```

```
100  LET S$ = "ZZZZZZZZZZZZZZZ"
110  FOR J = 1 TO N
120  IF S$ < = A$(J) THEN 150
130  LET S$ = A$(J)
140  LET J1 = J
150  NEXT J
160  PRINT A$(J1)
170  LET A$(J1) = "ZZZZZZZZZZZZZZZ"
180  NEXT I
999  END
```
(8-14)

Esaminiamo le fasi operative del programma, il cui diagramma di flusso è rappresentato in Figura 8-2. Gli statements 20-80 provvedono all'introduzione della lista di nomi in ordine arbitrario, poi sono avviati due cicli, l'uno inserito nell'altro. Quello esterno va dallo statement 90 allo statement 180, quello interno dal 110 al 150. Nello statement 100 poniamo

S$ = "ZZZZZZZZZZZZZZZ" (8-15)

Assumiamo che il nome sia formato da un massimo di quindici lettere: quindi S$ rappresenta la più grande rappresentazione numerica di un nome. Vediamo ora il ciclo interno; è sostanzialmente uguale a quello delle linee 150-170 del programma (6-12), nel quale si ricavava il valore numerico più basso. Ora, la variabile con il valore numerico più basso corrisponde al primo nome della lista alfabetica; quindi, terminata l'esecuzione del ciclo interno, J1 sarà il numero del nome che dovrà essere il primo nella lista alfabetica. Vale a dire che A$ (J1) sarà il primo nome della lista alfabetica. Dopo che il controllo lascia il ciclo interno, viene stampata A$(J1). Poi poniamo

A$(J1) = "ZZZZZZZZZZZZZZZ" (8-16)

ed attiviamo di nuovo il ciclo. Il primo nome della lista alfabetica è diventato l'ultimo, per cui, tornando in funzione il ciclo interno, verrà stampato il secondo nome della lista alfabetica. Di conseguenza, al termine del ciclo esterno, avremo tutta la lista stampata in ordine alfabetico.
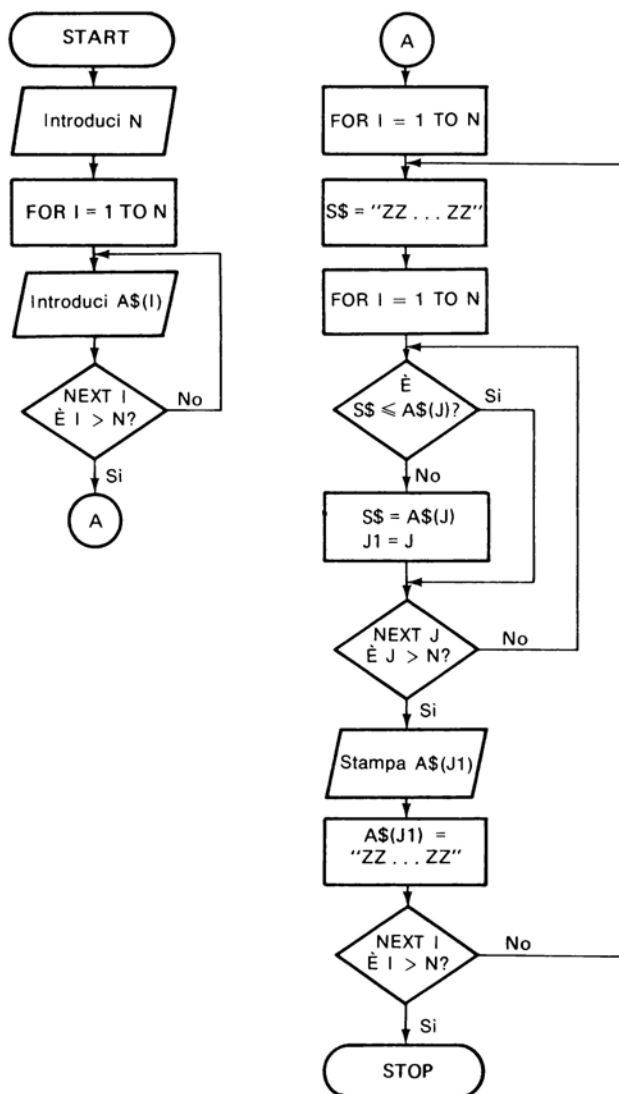
Figura 8.2 — Diagramma di flusso del programma (8-14) che mette in ordine alfabetico una lista di nomi.

8-3. USO DELLE FUNZIONI E DELLE SUBROUTINES CON LE VARIABILI-STRINGA

Le funzioni e le subroutines si possono usare con le variabili-stringa. Nelle subroutines non devono comparire nomi che le individuano né variabili fittizie. Quindi,

quando si usano con variabili-stringa, la procedura è sostanzialmente la stessa di quella usata con le normali variabili numeriche.

Nel caso delle funzioni, occorre badare particolarmente a che la lista di variabili che compare nella funzione abbia *forma identica* alla lista che compare nello statement di richiesta. Vediamo ad esempio il seguente segmento di programma:

```
10   DEF FNA(A,B,C$)
20   LET FNA = 0
30   IF C$ = "LIBRO" THEN 50
40   LET FNA = A+B
50   LET FNA = FNA+2*A
60   FNEND

     .

100  LET Y = FNA (X,Z,B$)
```

(8-17)

La lista che compare nel nome della funzione è costituita, nell'ordine, da due valori numerici ed una variabile-stringa; quindi anche la lista di variabili che compare nello statement di richiesta dev'essere costituita da due variabili numeriche ed una variabile-stringa, nel medesimo ordine. Pertanto la lista di variabili fittizie e la lista che compare nello statement di richiesta deve avere esattamente lo stesso numero di termini, e la forma dei termini corrispondenti dev'essere dello stesso tipo. Ad esempio, se la terza variabile della lista delle variabili fittizie è una variabile-stringa, la terza variabile della lista dello statement di richiesta dev'essere una variabile-stringa, e così via.

Consideriamo la funzione che compare nella sequenza di programma (8-17): se B$ dello statement di richiesta è uguale a "LIBRO", il valore di Y è 2*X; se invece il valore di B$ non è "LIBRO", Y sarà uguale ad X+Z+2*X.

Una funzione ha sempre un valore. Ad esempio, nello statement 40 calcoliamo il valore numerico di FNA. Finora abbiamo considerato il valore della funzione come numerico, ma non è necessariamente così: può essere anche una stringa. Se il valore della funzione è una stringa, questa dev'essere citata in modo particolare: il nome della funzione dev'essere formato dalle due lettere FN seguite da un'altra lettera, seguita a sua volta dal simbolo $. Ad esempio,

```
10   DEF FNB$(A,B$,C$)
20   IF A = 0 THEN 50
30   LET FNB$ = B$
40   GO TO 60
```

```
 50  LET FNB$ = C$
 60  FNEND
      .
100  LET Q$ = "ERRATO"
110  LET R$ = "ESATTO"
120  LET S = 0
130  LET D$ = FNB$(S,Q$,R$)
```

(8-18)

Qui, se A ≠ 0, il valore di FNB$ ritornato al programma principale sarà lo stesso del valore assegnato a B$; se invece A = 0, il valore di FNB$ ritornato al programma principale è il valore assegnato a C$. Ad esempio, l'esecuzione del segmento di programma (8-18) darà D$ = "ESATTO".

8-3-1. Gli statements di CHANGE

Sono due statements che si possono usare con le variabili-stringa, peraltro non disponibili in tutte le versioni di BASIC. Si è già detto prima che le lettere vengono memorizzate mediante un codice numerico: orbene, si può fare in modo che il calcolatore assegni i valori numerici di questo codice a delle variabili. I valori numerici si possono impiegare, ad esempio, per scegliere tutte le parole che comincino con una determinata lettera, o per comporre e decifrare codici segreti, etc.: in quest'ultimo caso, per cambiare i numeri potremmo usare delle espressioni matematiche; di conseguenza cambierebbero le lettere stampate, ed otterremmo un codice cifrato.

Un codice numerico standard (con cifrato) è il cosiddetto codice ASCII a 7 bit; è presentato nella Tabella 8.1.

Supponiamo di voler convertire una variabile-stringa nel suo codice ASCII: ogni lettera sarà memorizzata come numero a parte, e la serie completa dei numeri sarà memorizzata in un vettore. Lo statement che genera la conversione è

```
100  CHANGE A$ TO B
```

(8-19)

La forma di questo statement è: numero di statement, la parola-chiave CHANGE, il nome di una singola variabile-stringa, la parola TO, il nome del vettore. Si noti che non compaiono indici.

Esaminiamo l'uso dello statement di CHANGE.

Carattere	Codice	Carattere	Codice
0	48	N	78
1	49	Ø	79
2	50	P	80
3	51	Q	81
4	52	R	82
5	53	S	83
6	54	T	84
7	55	U	85
8	56	V	86
9	57	W	87
A	65	X	88
B	66	Y	89
C	67	Z	90
D	68		46
E	69	,	44
F	70	;	59
G	71	?	63
H	72	$	36
I	73	+	43
J	74	-	45
K	75	/	47
L	76	*	42
M	77	↑	94
		(spazio)	32

Tabella 8.1 — Il codice ASCII a 7 bit di alcuni caratteri BASIC.

```
 10  INPUT A$
 20  CHANGE A$ TO N
 30  FOR I = 0 TO 10
 40  PRINT N(I)
 50  NEXT I
999  END                                              (8-20)
```

Si noti che stampiamo N(I) per I che va da 0 a 10, assumendo che in questo caso siano ammessi gl'indici zero. Allora N(1) sarà il codice ASCII della prima lettera di A$, N(2) sarà il codice ASCII della seconda lettera di A$, e così via. Il valore di N(0) sarà uguale al *numero di lettere* presenti in A$. L'esecuzione del programma potrà essere ad esempio questa:

```
? ABCD
4
65
```

```
66
67
68
0
0
0
0
0
0                                                        (8-21)
```

Supponiamo di aver introdotto ABCD. Nella stampa del vettore N, verrà stampato per primo N(0), che rappresenta il numero di lettere presenti nella stringa A$; sono poi stampati 65, 66, 67 e 68, che sono i rispettivi codici ASCII di A, B, C e D. Il resto del vettore N è costituito da zeri, che vengono pure stampati: se non vogliamo stampare gli zeri, possiamo modificare lo statement 30 come segue:

```
30   FOR I = 0 TO A(0)
```

allo scopo di stampare solo le prime cinque linee. Se comunque fosse stata introdotta una parola di sei lettere, verrebbero stampate sette linee, e così via.

Se intendiamo introdurre più di dieci lettere, dovremo dimensionare opportunamente il vettore N.

Si può anche fare il processo contrario, cioè convertire il vettore N in una variabile - stringa. Ci serviremo allora dello statement

```
70   CHANGE N TO A$
                                                         (8-22)
```

La sua forma è: numero di statement, la parola-chiave CHANGE, il nome del vettore senza indice, la parola TO, seguita dal nome della variabile-stringa. Si tenga presente che N è un vettore, mentre la variabile - stringa è una variabile singola.

8-3-2. Le funzioni di libreria relative alle variabili - stringa

Esistono due funzioni di libreria che si possono usare con le variabili-stringa. Sono simili allo statement di CHANGE, ma operano solo su singole lettere o numeri.

La prima fornisce il codice ASCII di una lettera, un numero o un simbolo, ed ha la forma

```
20   LET B = ASC(X)                                      (8-23)
```

Il nome della funzione è ASC. Si noti la lettera X che compare fra parentesi nello statement (8-23): essa *non è una variabile*, cioè B sarà uguale ad 88, che è l'equivalente in ASCII della lettera X. Analogamente,

```
30  LET Y = ASC( )                                        (8-24)
```

farà assegnare ad Y il valore 32, che è il codice ASCII dello spazio.

La seconda delle due funzioni è l'inverso della prima, in quanto converte un numero nel suo equivalente ASCII. Il nome della funzione è CHR$. Ad esempio,

```
50  LET A$ = CHR$(88)                                     (8-25)
```

farà assegnare "X" ad A$. Con questa funzione si può usare non solo un numero, ma anche una variabile. Ad esempio,

```
10  LET N = 88
20  LET A$ = CHR$(N)                                      (8-26)
```

farà assegnare "X" ad A$.

Molte delle operazioni di cui si parla qui non sono standard su tutti i calcolatori. Il manuale di BASIC che correda il calcolatore utilizzato vi dirà quali potete usare, e se ci sono delle varianti di cui tener conto.

ESERCIZI

8-1. Scrivete un programma in BASIC in cui si introducano come dati un nome ed una serie di tre voti. Il risultato in uscita dovrà essere il nome e la media dei voti.

8-2. Descrivete le fasi operative del seguente programma in BASIC:

```
10  DIM A$(10Q),B$(100)
20  PRINT "INTRODURRE IL NUMERO TOTALE"
30  INPUT N
40  PRINT "INTRODURRE I NOMI",
45  PRINT "PER PRIMO L'ULTIMO"
50  FOR I = 1 TO N
60  INPUT A1(I),B$(I)
70  NEXT I
80  FOR J = 1 TO N
```

```
 90 PRINT B$(J);A$(J)
100 NEXT J
110 END
```

Verificate i risultati ottenuti eseguendo il programma su di un calcolatore.

8-3. Descrivete le fasi operative del seguente segmento di programma scritto in BASIC:

```
 10 READ Z$,A,B$,D
 20 READ E$,F
 30 RESTORE*
 40 READ G$,H
 50 RESTORE$
 60 READ I$,J
 70 RESTORE*
 80 READ K$,L
     .
     .
     .
150 DATA "LIBRO",26,"47","CASA",98
999 END
```

Verificate i risultati ottenuti scrivendo un programma opportuno ed eseguendolo su di un clacolatore.

8-4. Scrivete un programma in BASIC che accetti una lista di parole e determini, per ciascuna parola, se è la parola DENARO.

8-5. Modificate il programma (8-13) nel modo seguente: dopo che il programma è stato eseguito una volta, vengono stampate le parole VUOI SAPERE LA MEDIA DI UN ALTRO STUDENTE? Se l'utilizzatore batte SI', la fase di "richiesta d'informazione" del programma verrà eseguita di nuovo; se l'utilizzatore batte NO, l'elaborazione termina.

8-6. Modificate il programma (6-12) in modo che si possa introdurre il nome di ciascuno studente. Nelle liste in uscita dovranno comparire i nomi degli studenti ed i loro numeri ID.

8-7. Modificate il programma dell'Esercizio 8-6 in modo che le liste vengano stampate in ordine alfabetico.

8-8. Scrivete un programma in BASIC che esegua su una scolaresca le seguenti operazioni: devono essere introdotti come dati il nome ed il numero ID di ciascuno studente, e, successivamente, per ogni numero ID di uno studente, i voti da esso riportati in quattro esercitazioni. Si devono calcolare le medie, Poi le medie, senza nessun altro dato, devono essere stampate in ordine de-

crescente, una per linea. Poi l'utilizzatore batterà i dati che rappresentano gli estremi numerici degl'intervalli delle lettere-media A, B, C, D ed F. Infine dovrà essere stampata una lista alfabetica che riporti il nome dello studente, la media numerica e la lettera-voto. Questi elementi dovranno formare una tabella con le seguenti intestazioni:

NOME NUMERO MEDIA VOTO

8-9. Modificate l'Esercizio 8-6 in modo che vengano stampati i nomi del primo e dell'ultimo studente.

8-10. Modificate il programma dell'Esercizio 8-8 in modo che, in seguito all'esecuzione del programma, venga stampata una serie di lettere, una per studente. Queste potranno ad esempio avere la forma

CARO ( )

IL SUO VOTO IN CALCOLATORE II È STATO------

DISTINTI SALUTI
IL PROFESSOR ROSSI

Al posto di ( ) dovrà comparire il nome dello studente. I trattini stanno ad indicare un testo, che varierà a seconda del voto, nel modo seguente:

A RISULTATO ECCELLENTE CONGRATULAZIONI
B LAVORO OTTIMO
C DOVREBBE RIVEDERE UN PO' L'ELABORATO
D MEDIOCRE, RIVEDA L'ELABORATO E NE PARLI CON ME
F INSUFFICIENTE, RIPETA IL CORSO

8-11. Scrivete un programma che accetti una lista di parole e in seguito stampi tutte quelle che cominciano con E.

8-12. Scrivete un programma che costruisca e decifri un codice cifrato. Il codice dovrà essere tale per cui A sia sostituita da C, B da E, ..., Z da B. Per la codifica e la decodifica usate espressioni aritmetiche.

CAPITOLO 9

MESSA A PUNTO DEI PROGRAMMI

Scrivendo un programma, capita di commettere uno o più errori, chiamati in inglese bugs. La correzione di questi errori è detta *debugging*, o *messa a punto*, del programma: di questo parleremo in questo capitolo. È importante rendersi conto del fatto che tutti i programmatori commettono degli errori nello scrivere i loro programmi, quindi non scoraggiatevi se ce ne sono nel vostro. La messa a punto è un momento standard, molto importante, del processo di programmazione.

Gli errori che compaiono in un programma si possono dividere in due categorie: quelli della prima categoria sono individuabili in fase di compilazione; quelli della seconda si verificano in fase di esecuzione, dopo che nella compilazione non ne sono stati individuati. Molti calcolatori che lavorano in BASIC impiegano interpreti, invece dei compilatori: sia gl'interpreti che i compilatori "traducono" il programma scritto in BASIC nel linguaggio macchina. Si verifica allora che gli errori che verranno individuati in fase di compilazione se si usa il compilatore saranno invece individuati in fase di esecuzione se si usa l'interprete.

Gli errori individuabili in fase di compilazione sono quelli che, in qualche modo, trasgrediscono le regole del BASIC. Supponiamo ad esempio di aver scritto:

```
20   LET B = C(D+E)
```

Manca l'* fra C e la parentesi, e questo non è ammesso. Gli errori *di sintassi* come questo sono individuati dal compilatore, e di conseguenza viene stampato un messaggio di errore: in BASIC questo è costituito dal numero di statement più una breve frase che dà un'idea del tipo di errore.

Si hanno errori di esecuzione quando, pur essendo stato il programma compilato esattamente, non si ottengono i risultati desiderati. Questo può dipendere da un errore logico nell'algoritmo usato nel programma, oppure da un errore di stampa. Supponiamo ad esempio di aver intenzione di scrivere

```
10   LET A = B*C
```

e di scrivere invece

```
10  LET A = B+C
```

I due statements sono entrambi perfettamente corretti, ma il secondo porterà ad una risposta errata.

Un altro tipo di errore che può presentarsi durante l'esecuzione è il superamento per eccesso (overflow) o per difetto (underflow) della capacità aritmetica del calcolatore. Ad esempio, supponiamo di avere

```
40  LET A = B↑C
```

in cui si siano già fatte le assegnazioni B = 1E36 e C = 24736: all'esecuzione di questo statement si avrà overflow, per cui il risultato sarà errato.

Si tenga presente che spesso una semplice lettura del programma ci permette di scoprire e correggere molti errori, soprattutto quando si è raggiunta una certa esperienza. Ma molte volte non si possono eliminare tutti gli errori in questo modo. In questo capitolo prenderemo in esame altre tecniche di messa a punto. Parleremo dapprima degli errori che vengono individuati in fase di compilazione.

Quando correggiamo un programma, dobbiamo modificare uno o più statements: quando il programma è eseguito in modalità BASIC, per modificare uno statement è sufficiente ribattere lo statement in questione, compreso il numero di statement. Il programma così corretto può essere salvato (v. par. 1-4). Il manuale di BASIC annesso al calcolatore utilizzato vi dirà l'esatto comando richiesto per salvare un programma corretto.

9-1. CORREZIONE DEGLI ERRORI INDIVIDUATI IN FASE DI COMPILAZIONE

Se uno statement o un gruppo di statements trasgrediscono le regole del BASIC, il programma non può essere compilato: in questo caso verrà stampato un messaggio di errore che, normalmente, indicherà il numero dello statement che fa nascere il problema, ed il tipo di errore. I messaggi di errore presentano delle differenze da un compilatore all'altro, ma i principi di base restano gli stessi.

Vediamo alcuni errori tipici. In realtà un messaggio di errore può avere un testo molto lungo. Noi cercheremo solo di darvi un'idea del tipo di errore. Se gli errori si presentano in fase di compilazione, il messaggio di errore sarà sempre preceduto da un punto interrogativo.

Gli errori più comuni sono gli errori di stampa. Ad esempio, se nel programma compare lo statement

```
40   LEG Y = B+2                                        (9-1)
```

sarà stampato un messaggio di errore di questo tipo:

```
?UNRECOGNIZABLE STATEMENT IN LINE 40                    (9-2)
```

Controllando lo statement 40, vedremo immediatamente che LET è stato scritto male.

Uno statement come

```
15   LET YY = A+B                                       (9-3)
```

provocherà un messaggio di errore che dirà che nella linea 15 c'è una variabile non lecita: si ricordi infatti che le variabili in BASIC devono essere costituite o da una sola lettera, o da una lettera seguita da un numero, o, nel caso delle variabili-stringa, da una lettera seguita dal simbolo $.

Un altro errore molto comune è quello di omettere l'asterisco fra due paia di parentesi:

```
20   LET A = (B+C)(D+E)                                 (9-4)
```

In questo caso verrà stampato un messaggio di errore che segnalerà la presenza nella linea 20 di un formato errato. Con alcuni compilatori si ha un messaggio di errore generico, tipo

```
?ILLEGAL SYNTAX IN LINE 20
```

Altri messaggi di errore, più specifici, segnalano che manca un certo simbolo laddove era previsto, come ad esempio l'asterisco, o la barra, fra due paia di parentesi.

Un altro errore che si fa comunemente è dimenticare d'inserire lo statement di NEXT quando s'imposta un ciclo. Vediamo ad esempio la sequenza

```
150   FOR I = 1 TO N
160   LET A(I) = A(I)*Z
170   PRINT A(I)
999   END                                               (9-5)
```

Manca lo statement NEXT I, e sarà quindi generato un messaggio di errore.

Anche la presenza di variabili diverse negli statements di FOR e NEXT dà luogo ad un messaggio di errore. Ad esempio,

```
150  FOR I = 1 TO N
160  PRINT A(I)
170  NEXT J
999  END
```
(9-6)

Lo statement 170 dovrebbe contenere NEXT I, e non NEXT J, per cui sarà stampato un messaggio di errore.

Nel caso dell'esempio (9-5), il messaggio di errore segnalerà che è presente uno statement di FOR senza il relativo statement di NEXT; nel caso dell'esempio (9-6), a seconda del sistema BASIC, potremo avere due tipi di messaggio di errore: il primo segnalerà che vi è uno statement di FOR senza il relativo statement di NEXT, il secondo che lo statement di NEXT non ha il relativo statement di FOR.

Abbiamo visto alcuni errori tipici che vengono individuati in fase di compilazione. È chiaro che ce ne sono molti altri. In genere il messaggio di errore dà indicazioni sufficienti sul problema perché gli errori si possano correggere facilmente.

9-2. CORREZIONE DEGLI ERRORI CHE SI PRESENTANO IN FASE DI ESECUZIONE

Anche se in un programma non ci sono errori che ne impediscono la compilazione, si possono ugualmente avere risultati errati: ad esempio può verificarsi un superamento della capacità aritmetica, perché i numeri diventano troppo grandi o troppo piccoli. A volte il superamento per eccesso è provocato dal fatto che i dati in ingresso sono tali per cui il numero che si ottiene come risultato eccede l'intervallo di numeri su cui il calcolatore è in grado di operare: in questo caso non si ha un errore vero e proprio. Altre volte invece il problema nasce dalla limitatezza della forma del programma. Supponiamo ad esempio di voler eseguire il calcolo

$$x = \left(\frac{ab}{a-b} + b \right) (a-b) \qquad (9\text{-}7)$$

Il seguente statement:

```
30 X = (A*B/(A-B)+B)*(A-B)
```
(9-8)

è in grado di calcolare x.

Ora supponiamo che A e B siano uguali. All'esecuzione dello statement (9-8), prima si moltiplicherà A per B, poi si dividerà il prodotto per A–B: in questo caso la divisione è fatta per zero, e perciò non si può ottenere una risposta. In realtà, anche

se A e B non fossero uguali, ma fossero numeri molto grandi e pressoché uguali, si avrebbe un superamento per eccesso, e di conseguenza il calcolo non sarebbe corretto. Ma i problemi di questo tipo possono essere evitati. Ad esempio, possiamo modificare l'espressione, ottenendo

$$x = ab + b(a-b) \tag{9-9}$$

Lo statement BASIC corrispondente è

```
30   LET X = A*B+B*(A-B)                                          (9-10)
```

Ora non si divide per la differenza di due numeri, e di conseguenza il problema è eliminato.

Ma non tutti i problemi di overflow si eliminano con tanta facilità: comunque, se capita un problema di questo tipo, con un opportuno trattamento delle equazioni si riesce ad eliminarlo.

Altri errori possono derivare da errori di stampa. Supponiamo ad esempio di voler ricavare la media degli elementi di un vettore, e poi dividere ciascun elemento per la media ottenuta: per far questo scriviamo il seguente segmento di programma:

```
 30   REM SEGMENTO DI PROGRAMMA CON
 35   REM UN ERRORE DI STAMPA
 40   LET N = 10
 50   LET S = 0
 60   FOR I = 1 TO N
 70   LET S = S*A(I)
 80   NEXT I
 90   LET S = S/N
100   FOR J = 1 TO N
110   A(J) = A(J)/S
120   NEXT J                                                      (9-11)
```

Lo statement 70 dovrebbe essere

```
70   LET S = S+A(I)                                               (9-12)
```

Così com'è, S sarà sempre uguale a zero, e quindi, all'esecuzione dello statement 110, si avrà un caso di overflow. Con alcuni calcolatori, se si prova a dividere per zero, l'elaborazione s'interrompe: in questo caso si parla di *controllo sulla divisione*. In tutti e due i casi, il calcolo non può proseguire.

Quando ci si trova di fronte ad un problema di questo tipo, è possibile individuare

l'errore con un'accurata rilettura del programma. Nel caso di errori di stampa, il sistema spesso continua a funzionare.

Tuttavia molte voltge l'errore è *logico*, cioè è un errore d'impostazione dell'algoritmo. Supponiamo ad esempio di scrivere il segmento del programma che ricava la media, di cui si è appena parlato, nel modo seguente:

```
 40  REM PROGRAMMA CON UN ERRORE
 45  REM LOGICO ELEMENTARE
 50  LET S = 0
 60  FOR I = 1 TO N
 70  LET S = A(I)
 80  NEXT I
 90  LET S = S/N
100  FOR J = 1 TO N
110  A(J) = A(J)/S
120  NEXT J                                        (9-13)
```

Il programma ora sarà eseguito senza problemi; tuttavia, S sarà sempre uguale ad A(N)/N, ed otterremo delle risposte errate, perché lo statement 70 dovrebbe essere

```
70  LET S = S+A(I)
```

Supponiamo che non sia chiaro in quale punto del programma si trovi l'errore: in questo caso si possono inserire nel programma degli statements di PRINT di messa a punto. Questi statements visualizzeranno, passaggio per passaggio, il ciclo operativo. Ad esempio, nel segmento di programma (9-13) potremo inserire

```
72  REM STATEMENT DI MESSA A PUNTO
73  PRINT S                                        (9-14)
```

Ogni volta che il primo ciclo del programma compirà un giro, verrà stampato il valore di S, per cui si potrà facilmente vedere che S non è incrementato dell'opportuno valore di A(I) ad ogni giro del ciclo. E apparirà altrettanto chiaro che i valori successivi di S saranno uguali agli elementi del vettore A, e non alla loro somma. In tal modo ci renderemo conto che lo statement 70 è errato, e potremo quindi correggere l'errore.

In programmi complessi, può rendersi necessario inserire molti statements di PRINT, fino a quando l'errore, o gli errori, siano eliminati: eliminati gli errori, gli statements di PRINT usati per la messa a punto dovranno essere soppressi. Si tenga presente che, nella modalità BASIC, per far questo basta battere tutti i numeri di questi statements di PRINT e subito dopo il ritorno carrello.

Come regola generale, i risultati di un programma andrebbero verificati almeno

una volta con una calcolatrice, se è necessario, per accertarsi che il programma funzioni in modo esatto: se ci sono errori, si può provare a rifare passo passo con la calcolatrice le operazioni eseguite dal programma. Potrete inserire degli statements di PRINT per la messa a punto, per stampare tutti i risultati appropriati: è opportuno che questi statements visualizzino i singoli giri di un ciclo, etc. Il confronto di questi dati con i risultati ottenuti con la calcolatrice in genere mette in rilievo gli errori, per cui si può correggere il programma.

Vediamo un altro tipo di errore che si commette facilmente. Supponiamo di avere

```
10  DIM A(20)
20  INPUT M,N
30  T = M*N
40  FOR J = 1 TO T
50  LET A(J) = J
60  NEXT J                                    (9-15)
```

Può capitare che i valori introdotti di M e di N siano tali per cui T sia maggiore di 20, che è il valore al quale è stato dimensionato il vettore A.

In molti casi, verrà stampato un messaggio di errore di questo tipo:

```
IMPOSSIBLE ARRAY SIZE IN LINE 50
```

che vi ricorda che il vettore A non è stato dimensionato in modo da essere sufficientemente grande. Un altro errore frequente è quello di dimenticare di dimensionare un vettore. In BASIC i vettori sono automaticamente dimensionati a 10 (o 11) elementi; in ogni caso, se si cerca di aumentare la dimensione di un vettore oltre il valore per il quale è stato dimensionato, si possono avere degl'insoliti messaggi di errore, che segnalano un errore di locazione di memoria. Se vi capitano messaggi di questo genere, controllate le dimensioni dei vettori.

Ricordate che in quasi tutti i programmi intervengono degli errori. Se ce ne sono nei vostri programmi, non scoraggiatevi; la messa a punto è uno dei momenti costitutivi della scrittura dei programmi.

ESERCIZI

9-1. Scrivete l'elenco completo delle dichiarazioni di errore in BASIC previste dal calcolatore che utilizzate.

9-2. Scrivete un programma che contenga degli errori di stampa. Eseguite il programma ed esaminate i messaggi di errore.

9-3. Spiegate come da tecniche di programmazione limitate possa risultare un overflow.

9-4. Illustrate le procedure usate per individuare gli errori logici presenti in un programma.

9-5. Supponiamo che il seguente programma scritto in BASIC richieda l'introduzione di due vettori: elevate al quadrato gli elementi di entrambi, quindi moltiplicate gli elementi del primo per 2, e gli elementi del secondo per 3. Sommate poi i due vettori risultanti. Il risultato della somma dovrà essere stampato. Il programma ha degli errori. Verificate i risultati ottenuti eseguendo il programma su di un calcolatore. N dev'essere uguale massimo a 5.

```
 10  DIM A(8,8),B(8,8)
 20  INPUT N
 30  FOR I = 1 TO N
 40  FOR J = 1 TO N
 50  INPUT A(I,J),B(I,J)
 60  NEXT I
 70  FOR I = 1 TO N
 80  FOR J = 1 TO N
 90  LET A(I,J) = A(I,J)*2
100  LET B(I,J) = B(I,J)↑2
110  LET A(I,J) = A(I,J)*A(I,J)
120  LET B(I,J) = B(I,J)+B(I,J)
130  LET A(I,J) = A(I,J)+B(I,J)
140  NEXT J
150  NEXT I
160  FOR I = 1 TO N
170  FOR J = 1 TO N
180  PRINT A(I,J)
190  PRINT
200  NEXT I
210  NEXT J
222  END
```

9-6. Illustrate le fasi operative del programma seguente. Per quali valori delle quantità in ingresso si avranno risultati errati?

```
10  DIM A(100),B(100)
20  INPUT M,N
30  LET W = M*N
40  FOR J = 1 TO W
50  INPUT A(W),B(W)
60  LET A(W) = A(W)+B(W)
70  PRINT A(W)
80  NEXT W
90  END
```

CAPITOLO 10

LE OPERAZIONI SU VETTORI E MATRICI

Nel linguaggio BASIC esistono degli speciali statements che ci permettono di operare facilmente sulle matrici: è di tali statements che parleremo in questo capitolo. Le matrici hanno sostanzialmente la stessa forma dei vettori a due dimensioni, e il tipo di operazioni che illustreremo in questo capitolo si può applicare anche ai vettori.

Le matrici si usano spesso per rappresentare i coefficienti e le variabili di sistemi di equazioni. Se non avete familiarità con le matrici, potete saltare il capitolo; tuttavia le operazioni sulle matrici sono spiegate abbastanza chiaramente perché anche chi non ha pratica di questo argomento possa seguire il discorso. Dato che in BASIC le matrici ed i vettori in sostanza sono considerati come equivalenti, gli statements per le matrici che diamo qui sono utili anche se si lavora con vettori. Si può accentuare questa analogia: così, ad esempio, matrici e vettori vengono denominati allo stesso modo, cioè il nome di una matrice è costituito da una sola lettera. Gli elementi di una matrice, inoltre, si designano con indici, allo stesso modo degli elementi di un vettore: ad esempio, A(3,2) è l'elemento di una matrice posto sulla riga 3 della colonna 2. Ancora, le matrici si dimensionano allo stesso modo dei vettori; ad esempio,

```
10   DIM A(30),B(20,15)
```

serve a dimensionare due matrici, l'una formata solo da una colonna di 30 elementi, l'altra a due dimensioni, con 20 righe e 15 colonne. Se una matrice non è dimensionata, e consiste di una sola colonna, si assume che sia formata da 10 (o 11) elementi; analogamente si dà per scontato che una matrice a due dimensioni non dimensionata sia formata da 10 (o 11) righe e 10 (o 11) colonne.

Come regola generale, tutte le tecniche di trattamento dei vettori sono applicabili alle matrici; in particolare, in questo capitolo illustreremo degli statements utilizzabili sia con matrici che con vettori. Generalmente questi statements ci permettono di scrivere dei programmi più brevi; vedremo infatti che un solo statement specifico per le matrici può sostituire diversi statements "normali".

10-1. STATEMENTS D'INGRESSO PER MATRICI

Cominciamo la nostra trattazione con alcuni statements speciali che si possono usare vantaggiosamente quando è necessario introdurre dei dati nelle matrici o nei vettori. Vediamo dapprima le matrici formate solo da una colonna: le matrici ad una sola dimensione si chiamano *vettori*.

Sappiamo già come introdurre i dati in un vettore. Ad esempio, si può ricorrere al programma seguente:

```
 10  DIM A(7)
 30  FOR I = 1 TO 7
 40  READ A(I)
 50  NEXT I
 90  DATA 5,7,9,11,13,15,17
100  END                                    (10-1)
```

La procedura è la stessa di quella illustrata nel paragrafo 6-1 a proposito dell'introduzione dei dati in vettori ad una dimensione; comunque c'è uno statement specifico, lo statement di MAT READ, che elimina la necessità di ricorrere al ciclo. Ed ecco un esempio dell'uso di questo statement:

```
 10  DIM A(7)
 20  MAT READ A
 90  DATA 5,7,9,11,13,15,17
100  END                                    (10-2)
```

Si noti che il solo statement 20 sostituisce il ciclo del segmento di programma (10-1).

Vediamo ora la forma specifica dello statement per introdurre dei dati in un vettore:

```
20  MAT READ A
```

Lo statement è formato dal numero di statement, seguito dalle parole-chiave MAT READ, seguite a loro volta da un nome di vettore. Esaminiamo ora come questo statement è eseguito: per entrambi i segmenti di programma (10-1) e (10-2) saranno realizzate le seguenti assegnazioni:

A(1) = 5
A(2) = 7
A(3) = 9
A(4) = 11

```
A(5) = 13
A(6) = 15
A(7) = 17
```
(10-3)

Generalmente si dimensiona un vettore in modo che sia il più grande tra quelli di cui si può aver bisogno; tuttavia spesso si introducono dati in quantità tale che la dimensione reale del vettore è minore di quella definita dallo statement di DIM. Così ad esempio abbiamo visto molti programmi in cui l'ingresso aveva la forma

```
10 DIM A(100)
20 READ N
30 FOR I = 1 TO N
40 INPUT A(I)
50 NEXT I
```

(10-4)

In questo caso A è dimensionato a 100 elementi; tuttavia, quando si esegue il programma, la dimensione reale del vettore A è definita dal valore di N, che dev'essere, chiaramente, minore di 100. Orbene, si può definire la dimensione reale del vettore mediante lo statement MAT READ associato ad altri statements, come appare nel seguente esempio:

```
10 DIM A(100)
20 READ N
30 MAT READ A(N)
```

(10-5)

Questa procedura è valida per la maggior parte dei calcolatori.

La variabile posta fra parentesi dopo il nome del vettore nello statement di MAT READ specificherà la dimensione effettiva del vettore. In realtà, non è indispensabile scrivere una variabile per specificare la dimensione del vettore. Ad esempio potremo scrivere

```
50 MAT READ A(7)
```
(10-6)

per specificare che nel vettore A dovranno essere introdotti sette elementi.

Fra quanto si verifica nell'esempio (10-4) e ciò che accade nell'esempio (10-5)

c'è una differenza: pur entrando nel vettore in entrambi i casi N elementi, nell'esempio (10-5) il calcolatore tratterà il vettore come se fosse formato da soli N elementi, mentre nell'esempio (10-4) il vettore sarà trattato comunque come se avesse 100 elementi. Con molti compilatori BASIC gli elementi non introdotti sono considerati come zeri. Nel prossimo paragrafo esamineremo il significato di queste affermazioni.

Fin qui abbiamo preso in considerazione i vettori. Vediamo ora le matrici a due dimensioni. Anche in questo caso si usa lo statement di MAT READ. Così i due segmenti di programma

```
10 DIM A(4,3)
10 FOR I = 1 TO 4
30 FOR J = 1 TO 3
40 READ A(I,J)
50 NEXT J
60 NEXT I
   .
90 DATA 1,2,3,4,5,6,7,8,9,10,11,12                    (10-7)
```

e

```
10 DIM A(4,3)
20 MAT READ A
   .
90 DATA 1,2,3,4,5,6,7,8,9,10,11,12
                                                      (10-8)
```

sono equivalenti.

Per entrambi saranno realizzate le seguenti assegnazioni:

```
A(1,1) = 1     A(1,2) = 2     A(1,3) = 3
A(2,1) = 4     A(2,2) = 5     A(2,3) = 6
A(3,1) = 7     A(3,2) = 8     A(3,3) = 9
A(4,1) = 10    A(4,2) = 11    A(4,3) = 12             (10-9)
```

Siamo anche in grado di specificare che la grandezza della matrice in cui introdurre i dati è minore delle sue dimensioni. Ad esempio, i due segmenti di programma

```
10  DIM A(100,100)
20  READ M,N
30  FOR I  = 1 TO M
40  FOR J = 1 TO N
50  READ A(M,N)
60  NEXT J
70  NEXT I
```

(10-10)

e

```
10  DIM A(100,100)
20  READ M,N
30  MAT READ A(M,N)
```

(10-11)

sono analoghi.

I concetti di base sono sostanzialmente gli stessi che per i vettori. Si osservi che fra l'esempio (10-10) e l'esempio (10-11) esiste una differenza: con il programma (10-11) la matrice A sarà trattata come se fosse formata da M righe ed N colonne; con il programma (10-10) invece la matrice A sarà trattata come se fosse formata da 100 righe e 100 colonne. Nel prossimo capitolo ci soffermeremo sul significato di quanto si è appena detto.

10-1-1. Uso di più di una matrice

Alcune versioni di BASIC permettono di citare più di una matrice in uno statement di MAT READ. Ad esempio,

```
50  MAT READ A,B
```

Prima saranno letti gli elementi della matrice A, poi quelli della matrice B. Inoltre nello statement si possono inserire anche le grandezze delle matrici.

10-1-2. Lo statement di MAT INPUT

Lo statement d'ingresso dati da tastiera, usato con i vettori, è particolarmente vantaggioso. Questo statement ha la forma

```
30   MAT INPUT A                                        (10-12)
```

cioè numero di statement, le parole-chiave MAT INPUT, nome del vettore. Illustriamone l'uso.

```
10   DIM A(100)
20   MAT INPUT A                                        (10-13)
```

Quando si esegue il programma, sulla telescrivente apparirà un punto interrogativo (?), e di conseguenza l'utilizzatore introdurrà un certo numero di dati, separati da virgole: la quantità di dati introdotti dev'essere minore o uguale alla dimensione del vettore (o della matrice). Questi dati saranno assegnati nell'ordine agli elementi del vettore A. Se, ad esempio, in risposta al ?, l'utilizzatore batte

```
? 1,3,5,7,9 (ritorno carrello)
```

saranno assegnati i seguenti valori:

```
A(1) = 1
A(2) = 3
A(3) = 5
A(4) = 7
A(5) = 9
```

I rimanenti elementi del vettore non saranno interessati dall'operazione: ad esempio, con la maggior parte dei compilatori BASIC, si avrà A(6) = 0, ... A(100) = 0. Non appena viene premuto il ritorno carrello, l'introduzione di dati nel vettore A ha termine. In alcuni casi il numero dei dati è tale che essi non entrano su una sola riga di telescrivente: la presenza del segno & a fine riga indica che il ritorno carrello non segna la fine dell'ingresso dati. Allora il calcolatore risponderà con un punto interrogativo all'inizio della riga successiva. Ad esempio, le due modalità d'introduzione dati da terminale

```
? 1,3,-5,7,9,11,13& (ritorno carrello)
? 15,17,19 (ritorno carrello)
```

e

? 1,3,-5,7,9,11,13,15,17,19 (ritorno carrello) (10-14)

sono equivalenti.

A volte è opportuno sapere quanti elementi sono stati introdotti: la funzione di libreria BASIC NUM fornirà il numero di elementi introdotti in risposta ad uno statement di MAT INPUT. Supponiamo ad esempio di voler introdurre un numero arbitrario di elementi e ricavarne la media. Potremo allora scrivere il seguente segmento di programma:

```
10  REM PROGRAMMA DI MEDIA
20  DIM A(100)
30  MAT INPUT A
40  LET S = 0
50  LET B = NUM
60  FOR I = 1 TO B
70  LET S = S+A(I)
80  NEXT I
90  LET S = S/B
```

(10-15)

Nello statement 40 poniamo B = NUM: quando questo statement è eseguito, B sarà uguale al numero degli elementi che sono stati introdotti in risposta allo statement 30. Infatti, all'esecuzione dello statement 30, sulla telescrivente appare un ?, e, fintantoché i dati vengono introdotti, l'esecuzione del programma non va oltre questo statement. Quando si preme il tasto di ritorno carrello, l'esecuzione prosegue e, all'esecuzione dello statement 40, B sarà uguale al numero degli elementi appena introdotti.

Si noti che la forma MAT INPUT A(N) è *errata*, perché la grandezza del vettore non può venir specificata con lo statement di MAT INPUT.

Alcune versioni di BASIC permettono di usare lo statement di MAT INPUT non solo con i vettori, ma anche con le matrici a due dimensioni: con le matrici a due dimensioni è però svantaggioso, perché il numero di righe e di colonne dev'essere specificato nello statement di DIM. (Nel paragrafo 10-4 vedremo delle varianti a questo punto.) Ad esempio, il seguente segmento di programma:

```
10  DIM A(100,100)
20  FOR I = 1 TO 100
30  FOR J = 1 TO 100
```

```
40  INPUT A
50  NEXT J
60  NEXT I
```

è equivalente a questo:

```
10  DIM A(100,100)
20  MAT INPUT A
```

Gli elementi della matrice sono introdotti nell'ordine, esattamente come nell'esempio (10-14): i primi numeri assegnano i valori agli elementi della prima riga, e così via, cioè l'assegnazione dei dati avviene come negli esempi (10-8) e (10-9).

Si tenga presente che molte versioni di BASIC permettono l'uso dello statement di MAT INPUT solo con i vettori, e non con le matrici a due dimensioni. In questi casi, per introdurre da telescrivente i dati per una matrice ci si può sempre servire di statements associati ad una procedura ciclica: allora il numero di dati da introdurre va specificato sotto forma di dato. Ad esempio,

```
10  DIM A(100,100)
20  INPUT M,N
30  FOR I = 1 TO M
40  FOR J = 1 TO N
50  INPUT A(I,J)
60  NEXT J
70  NEXT I                                    (10-16)
```

La matrice sarà ancora considerata come costituita da 100 righe e 100 colonne. Con la maggior parte delle versioni di BASIC gli elementi che non vengono introdotti sono uguali a zero. Nel paragrafo 10-4 vedremo come si possa modificare la grandezza della matrice.

10-2. STATEMENTS DI USCITA PER MATRICI

Volendo avere in uscita i dati di una matrice, possiamo ricorrere allo statement di MAT PRINT. La sua forma è

```
50  MAT PRINT A                               (10-17)
```

cioè numero di statement, le parole-chiave MAT PRINT, nome della matrice. Inizialmente illustreremo la stampa dei vettori; vedremo poi le matrici a due dimensioni.

Illustriamo questo statement considerando il seguente semplice programma:

```
10   DIM B(8)
20   MAT READ B
30   MAT PRINT B
40   DATA 1,2,3,4,5,6,7,8
50   END
```
(10-18)

Si avrà in uscita il seguente risultato:

```
1
2
3
4
5
6
7
8
```

Cioè il vettore, che è una matrice costituita da una sola colonna, sarà stampato sotto forma di una singola colonna. Si tenga presente che si otterrebbero gli stessi risultati se, al posto dello statement 30, scrivessimo:

```
30   FOR I = 1 TO 8
32   PRINT B(I)
34   NEXT I
```
(10-19)

Ponendo una virgola alla fine dello statement di MAT PRINT, tutti i dati saranno stampati su una sola linea, se ci stanno. Se, ad esempio, nel programma (10-18) sostituiamo allo statement 30 quest'altro:

```
30   MAT PRINT B,
```
(10-20)

si avrà in uscita

```
1          2          3          4          5
6          7          8
```

Si tenga presente che si avrebbe lo stesso risultato in uscita se, nel segmento di programma (10-19), allo statement 32 sostituissimo lo statement seguente:

```
32  PRINT B(I),
```

Volendo ottenere una stampa più compatta occorre sostituire la virgola con un punto e virgola alla fine dello statement di MAT PRINT. Ad esempio, se nel programma (10-18) sostituiamo allo statement 30 lo statement

```
30  MAT PRINT B;                                         (10-21)
```

risulterà in uscita

```
1    2    3    4    5    6    7    8
```

Quando si usa lo statement di MAT PRINT con una matrice a due dimensioni, il risultato sarà stampato riga per riga. Consideriamo il programma seguente:

```
10  DIM A(2,3)
20  READ A
30  MAT PRINT A
40  DATA 1,2,3,4,5,6
50  END                                                  (10-22)
```

La matrice A è dimensionata in modo da avere 2 righe e 3 colonne; quindi il risultato in uscita sarà

```
1       2       3

4       5       6
```

Si noti la linea bianca fra le righe.

Vediamo ora che cosa accade quando i dati non entrano tutti su una stessa riga di stampa. Ad esempio, supponiamo di avere:

```
10  DIM A(3,8)
20  READ A
30  MAT PRINT A
40  DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
50  DATA 17,18,19,20,21,22,23,24
60  END                                                  (10-23)
```

Il risultato in uscita sarà

```
1       2       3       4       5
6       7       8

9       10      11      12      13
14      15      16


17      18      19      20      21
22      23      24
```

Si osservi che le righe della matrice sono evidenziate con le linee bianche. Pertanto la linea bianca non c'è fra la linea 1 e la linea 2 che, prese insieme, rappresentano la prima riga della matrice. A queste segue una linea bianca, dopo la quale ha inizio la seconda riga, e così via. Pertanto la matrice effettiva ha la forma

$$B = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \end{bmatrix} \qquad (10\text{-}24)$$

(Si osservi che le matrici sono rappresentate convenzionalmente racchiuse fra parentesi quadre, come si vede nel nostro esempio.)

Ponendo alla fine dello statement di MAT PRINT un punto e virgola, si ottiene una stampa più compatta. Ad esempio, se allo statement 30 del programma (10-23) sostituiamo lo statement

```
30   MAT PRINT A;
```

risulterà in uscita

```
1       2       3       4       5       6       7       8
9       10      11      12      13      14      15      16
17      18      19      20      21      22      23      24
```

Si tenga presente che con alcuni compilatori BASIC il punto e virgola fa sì che gli elementi delle colonne non siano stampati ordinatamente l'uno sotto l'altro.

In uno statement di MAT PRINT si può definire più di un vettore e/o matrice. Ad esempio, nella maggior parte delle versioni di BASIC le forme

```
30   MAT PRINT A,B                                        (10-25a)
```

e

```
30  MAT PRINT A,
31  MAT PRINT B                                    (10-25b)
```

sono equivalenti.

Sono equivalenti anche

```
40  MAT PRINT A,B,                                 (10-26a)
```

e

```
40  MAT PRINT A,
41  MAT PRINT B,                                   (10-26b)
```

come pure

```
50  MAT PRINT A;B;                                 (10-27a)
```

e

```
50  MAT PRINT A;
51  MAT PRINT B;                                   (10-27b)
```

Quando si usano gli statements di MAT PRINT, viene stampato l'intero vettore, o l'intera matrice. Consideriamo ad esempio il programma seguente:

```
10  DIM A(100)
20  MAT INPUT A
30  MAT PRINT A
40  END
```

Se noi introduciamo i dati

```
? 1,2,3,4
```

all'esecuzione del programma saranno stampati i seguenti dati:

```
1
2
```

```
3
4
0
0
0
.
0
```

Vale a dire che saranno stampati tutti i 100 elementi, con degli zeri per i dati non specificati: se vogliamo che vengano stampati solo i dati desiderati, possiamo usare la funzione NUM (v. par. 10-1) associata ad un ciclo e ad un normale statement di PRINT.

Usando lo statement di MAT READ, la grandezza del vettore può essere specificata nel corso dell'esecuzione. Ad esempio,

```
10   DIM A(100)
20   READ N
30   MAT READ A(N)
40   MAT PRINT A
50   DATA 4,1,2,3,4
60   END
```

Qui ad N è assegnato il valore 4. All'esecuzione dello statement di MAT READ, il calcolatore sarà informato che il vettore A è costituito da quattro soli elementi, e, di conseguenza, all'esecuzione dello statement di MAT PRINT, saranno stampati solo quattro elementi, vale a dire che gli zeri verranno soppressi. Con qualche versione di BASIC si può specificare la grandezza del vettore da stampare mediante lo stesso statement di MAT PRINT. Ad esempio, con

```
20   MAT PRINT A(N)
```

saranno stampati soltanto N termini.

Abbiamo illustrato questi concetti (riguardanti la quantità dei dati da stampare) riferendoci ai vettori, ma essi valgono anche per le matrici. Ad esempio, il programma seguente:

```
10   DIM A(20,20)
20   READ A(2,3)
30   MAT PRINT A
```

```
40  DATA 1,2,3,4,5,6
50  END
```

avrà in uscita gli stessi risultati dell'esempio (10-22).

10-3. OPERAZIONI ARITMETICHE SULLE MATRICI

È possibile fare specifiche operazioni aritmetiche sulle matrici. In questo paragrafo definiremo queste operazioni ed illustreremo gli statements del BASIC che si usano per realizzarle.

10-3-1. Somma di matrici

Quando si sommano due matrici, esse devono avere lo stesso numero di righe e lo stesso numero di colonne. La somma di due matrici è una matrice i cui elementi sono la somma degli elementi corrispondenti delle matrici di partenza. Ad esempio, se

$$A = \begin{bmatrix} 2\ 3\ 4 \\ 5\ 6\ 7 \end{bmatrix} \tag{10-28a}$$

e

$$B = \begin{bmatrix} 4\ 9\ 6 \\ 8\ 9\ 2 \end{bmatrix} \tag{10-28b}$$

risulterà

$$A + B = \begin{bmatrix} 2+4 & 3+9 & 4+6 \\ 5+8 & 6+9 & 7+2 \end{bmatrix} = \begin{bmatrix} 6 & 12 & 10 \\ 13 & 15 & 9 \end{bmatrix} \tag{10-29}$$

Per sommare due matrici si possono usare dei normali statements BASIC. Ad esempio, si otterrà lo scopo con il seguente segmento di programma:

```
10   FOR I = 1 TO 2
20   FOR J = 1 TO 3
30   LET C(I,J) = A(I,J)+B(I,J)
40   NEXT J
50   NEXT I                                   (10-30)
```

C'è tuttavia uno statement relativo alle matrici, che ci permette di eseguire quest'operazione con un minor numero di statements: si tratta dello statement

```
10   MAT C = A+B                              (10-31)
```

Chiaramente A, B e C devono essere matrici delle stesse dimensioni. (Se è necessario, saranno dimensionate.) Si può scrivere anche

```
10   MAT A = A+B                              (10-32)
```

Ciò vuol dire che il valore memorizzato nella locazione A(I,J) sarà la somma del vecchio valore ivi memorizzato più il valore memorizzato nella locazione B(I,J).

Il seguente statement invece

```
20   MAT D = A+B+C                            (10-33)
```

è errato.

Le operazioni aritmetiche sulle matrici si possono eseguire solo con *non più di* una operazione alla destra del segno uguale: volendo sommare tre matrici, dovremo ricorrere a due statements. Ad esempio,

```
20   MAT D = A+B
30   MAT D = D+C                              (10-34)
```

Come risultato D sarà la somma delle matrici A, B e C.

I concetti che regolano la somma di matrici sono applicabili senza alcuna modifica ai vettori.

10-3-2. Sottrazione di matrici

I concetti che regolano la sottrazione di matrici discendono direttamente da quelli relativi alla somma di matrici. Ad esempio, sottraendo le matrici dell'esempio (10-28), abbiamo

$$A - B = \begin{bmatrix} 2-4 & 3-9 & 4-6 \\ 5-8 & 6-9 & 7-2 \end{bmatrix} = \begin{bmatrix} -2 & -6 & -2 \\ -3 & -3 & 5 \end{bmatrix} \qquad (10\text{-}35)$$

La sottrazione di matrici si può realizzare semplicemente con uno statement di MAT. Ad esempio,

```
20   MAT C = A-B
```
(10-36)

È chiaro che A e B devono essere delle matrici.

Alla destra del segno uguale può comparire una sola operazione. Volendo eseguire l'operazione matriciale A-B+C-D, potremo ricorrere alla seguente serie di statements:

```
10   MAT E = A-B
30   MAT E = E+C
40   MAT E = E-D
```
(10-37)

10-3-3. Moltiplicazione scalare

A volte occorre moltiplicare ciascun elemento di una matrice per uno stesso numero: si parla in questo caso di moltiplicazione scalare. Se ad esempio moltiplichiamo ciascun elemento della matrice A dell'esempio (10-28a) per 2, otteniamo

$$\begin{bmatrix} 4 & 6 & 8 \\ 10 & 12 & 14 \end{bmatrix} \qquad (10\text{-}38)$$

Lo statement del BASIC che esegue la moltiplicazione scalare è:

```
20   MAT B = (2)*A
```
(10-39)

La forma di questo statement è la seguente: numero di linea, la parola MAT, un nome di matrice, il segno uguale, un numero o un'espressione numerica posti fra parentesi, asterisco, la matrice da moltiplicare per la costante. Al posto del moltiplicatore costante può comparire una variabile o un'espressione. Ad esempio,

```
30   MAT D = (K)*B
```
(10-40)

oppure

```
40   MAT F = (2*J+K↑2)*D
```
(10-41)

Si tenga presente che J e K devono essere numeri, non matrici, mentre B, D ed F devono essere matrici.

Anche la seguente espressione:

```
20   MAT A = (K)*A
```
(10-42)

è corretta.

Allora, in seguito all'esecuzione di questo statement, i valori memorizzati per ciascun elemento della matrice A saranno uguali a K volte i rispettivi valori memorizzati in precedenza.

10-3-4. Moltiplicazione di matrici

Due matrici possono essere moltiplicate; ma si tratta di una moltiplicazione diversa da quella ordinaria. Innanzitutto l'ordine del prodotto è importante. Perché la moltiplicazione di matrici si possa fare, il numero delle colonne della prima matrice dev'essere uguale al numero delle righe della seconda. Realizzata questa condizione, possiamo definire la moltiplicazione di matrici con la seguente equazione:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{k1} & a_{k2} & \dots & a_{kn} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1j} \\ b_{21} & b_{22} & \dots & b_{2j} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nj} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1j} \\ c_{21} & c_{22} & \dots & c_{2j} \\ \dots & \dots & \dots & \dots \\ c_{k1} & c_{k2} & \dots & c_{kj} \end{bmatrix} \qquad (10\text{-}43a)$$

dove

$$c_{ir} = a_{i1}b_{1r} + a_{i2}b_{2r} + \dots + a_{in}b_{nr} \qquad (10\text{-}43b)$$

Illustriamo le operazioni.

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 1 & 3 & 7 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 4 & 1 \\ 3 & 6 \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix} = \begin{bmatrix} 18 & 22 \\ 36 & 44 \\ 34 & 47 \end{bmatrix} \qquad (10\text{-}44)$$

dove

$$c_{11} = 1(1) + 2(4) + 3(3) = 18$$
$$c_{12} = 1(2) + 2(1) + 3(6) = 22$$
$$c_{21} = 2(1) + 4(4) + 6(3) = 36$$
$$c_{22} = 2(2) + 4(1) + 6(6) = 44$$
$$c_{31} = 1(1) + 3(4) + 7(3) = 34$$
$$c_{32} = 1(2) + 3(1) + 7(6) = 47$$

Si noti che la matrice risultante ha il numero di righe della prima matrice ed il numero di colonne della seconda. In BASIC è molto semplice ottenere il prodotto di matrici; basta scrivere:

```
40   MAT C = A*B
```
(10-45)

Ad esempio, se

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 1 & 3 & 7 \end{bmatrix}$$

e

$$B = \begin{bmatrix} 1 & 2 \\ 4 & 1 \\ 3 & 6 \end{bmatrix}$$

come risultato dello statement (10-45) avremo:

$$C = \begin{bmatrix} 18 & 22 \\ 36 & 44 \\ 34 & 47 \end{bmatrix}$$

Non dimenticate che, affinché la matrice prodotto esista, il numero di colonne della prima matrice dev'essere uguale al numero di righe della seconda. Come regola generale, se A e B sono matrici, $A*B \neq B*A$: infatti, anche se A*B esiste, non è detto che esista B*A.

Volendo calcolare A*B*C, dovremo ricorrere a due distinti statements BASIC. Ripetiamo che, quando si ha a che fare con operazioni matriciali, alla destra del segno uguale deve comparire una sola operazione. Quindi, per realizzare il prodotto di tre matrici, scriveremo:

```
10   MAT D = A*B
20   MAT C = D*B
```
(10-46)

Si tenga presente che l'ordine delle operazioni è importante.

Se una matrice comprende un egual numero di righe e di colonne, la si può moltiplicare per sé stessa. Ad esempio, il seguente statement:

```
50   MAT G = X*X
```
(10-47)

è corretto.

Invece l'operazione seguente:

```
40   MAT A = A*B
```

non è valida nel caso di moltiplicazione matriciale, in quanto, nelle moltiplicazioni di matrici, non possiamo avere una stessa matrice sia alla sinistra che alla destra del segno uguale.

10-3-5. Uguaglianza di matrici

Due matrici sono uguali quando gli elementi corrispondenti sono uguali, e ciascuna delle due ha lo stesso numero di righe e lo stesso numero di colonne. Ad esempio, le due matrici seguenti:

$$\begin{bmatrix} 2 & 4 & 6 \\ 1 & 3 & 7 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 6 \\ 1 & 3 & 7 \end{bmatrix}$$

sono uguali.

Si possono assegnare i valori di una matrice in modo che l'intera matrice sia uguale ad un'altra matrice. Per questo useremo lo statement BASIC

```
20   MAT B = Y
```
(10-48)

Con esso a ciascun elemento della matrice B sarà assegnato un valore uguale al valore dell'elemento corrispondente della matrice Y.

In questo paragrafo siamo partiti dal presupposto che tutte le matrici fossero dimensionate correttamente, cioè che, se sommiamo delle matrici, esse siano dimensionate in modo che il numero delle loro righe (colonne) sia uguale, e così via. Negli esempi ci siamo serviti di matrici, ma il discorso vale ugualmente per i vettori. Si

tenga presente che, nelle moltiplicazioni di matrici, la seconda matrice può essere un vettore, ma la prima no, benché quest'ultima possa essere una matrice con una sola riga: si noti che una matrice con più di una riga può moltiplicare un vettore.

10-4. OPERAZIONI MATRICIALI SPECIALI

Esistono parecchi statements BASIC speciali relativi alle matrici, che sono di grande utilità. Se ne parlerà in questo paragrafo.

10-4-1. Lo statement di MAT - ZER

Molte volte occorre specificare che tutti gli elementi di una matrice sono zero. Lo statement BASIC che esegue questa operazione è

```
20  MAT B = ZER                                   (10-49)
```

Cioè, porre uguale a zero una matrice signifca porre uguali a zero tutti i suoi elementi. Più in dettaglio, supponiamo di avere

```
10  DIM B(20,40)

        .

50  MAT B = ZER                                   (10–50)
```

B sarà una matrice formata da 20 righe e 40 colonne, i cui elementi saranno tutti uguali a zero. Spesso si vuol specificare che la grandezza della matrice sia minore di quella definita dallo statement di DIM; ad esempio, nel paragrafo 10-2, parlando della stampa dei vettori, si è detto che questo tipo di specificazione sarebbe auspicabile. Analogamente nel caso delle operazioni matriciali. Per specificare la grandezza di una matrice possiamo usare lo statement di ZER. Ad esempio,

```
10  DIM B(20,40)
20  INPUT M,N
30  MAT B = ZER(M,N)                              (10-51)
```

B sarà ora considerata come una matrice di M righe ed N colonne, con tutti gli elementi uguali a zero; di conseguenza in tutte le operazioni future B sarà trattata come una matrice di M righe ed N colonne. Naturalmente M ed N devono essere ugua-

li o minori delle dimensioni specificate. Si tenga presente che un successivo statement può cambiare la grandezza della matrice; ad esempio,

```
 10   DIM B(20,40)
        .
 50   MAT B = ZER(8,10)
        .
100   MAT B = ZER(12,15)
```

(10-52)

Fino allo statement 50, B sarà considerata una matrice 20 × 40; tra lo statement 50 e lo statement 100, sarà considerata formata da 8 righe e 10 colonne; dallo statement 100 in poi sarà considerata di 12 righe e 15 colonne. Si tenga sempre presente che la grandezza di una matrice non può mai essere maggiore delle sue dimensioni.

10-4-2. Lo statement di MAT-CON

È molto simile allo statement di MAT-ZER, solo che qui tutti gli elementi della matrice vengono posti uguali ad uno. Una forma tipica è

```
20   MAT C = CON
```

(10-53)

Tutti gli elementi della matrice C saranno ora uguali ad uno. Lo statement di MAT-CON si può usare per specificare la grandezza di una matrice, esattamente come lo statement di MAT-ZER. Ad esempio,

```
10   DIM C(100)
       .
50   MAT C = CON(20)
```

(10-54)

Qui C è un vettore dimensionato a 100 elementi. All'esecuzione dello statement 50, C sarà considerato formato da 50 elementi, tutti uguali ad uno.

10-4-3. Lo statement di MAT - IDN

Un tipo di matrice che si utilizza spesso è la matrice d'identità. Prima di esaminarla, definiamo alcuni termini. Si dice matrice *quadrata* una matrice che ha lo stesso numero di righe e di colonne. Il numero di righe e di colonne di una matrice quadrata è detto *ordine* della matrice. Si chiama *diagonale principale* di una matrice quadrata la diagonale che va dall'angolo in alto a sinistra all'angolo in basso a destra della matrice stessa. Ad esempio, una matrice quadrata di ordine tre è la seguente:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

I termini posti sulla diagonale principale sono a_{11}, a_{22} e a_{33}.

La matrice d'identità è definita nel modo seguente: una matrice quadrata i cui elementi sono tutti uguali a zero, tranne quelli posti sulla diagonale principale, che sono tutti uguali ad uno. Ad esempio una matrice d'identità di ordine tre è la seguente:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (10\text{-}55)$$

La matrice d'identità ha alcune interessanti proprietà; ad esempio, se A è una matrice quadrata ed I è la matrice d'identità, e sono dello stesso ordine, allora

$$A*I = I*A = A \qquad (10\text{-}56)$$

Se vogliamo porre una matrice uguale alla matrice d'identità, ricorriamo allo statement

```
10   DIM C(20,20)
20   MAT C = IDN                                    (10-57)
```

Ora C sarà uguale ad una matrice d'identità di ordine 20. L'ordine della matrice d'identità può essere inferiore alla sua dimensione. Ad esempio, si può scrivere

```
10   DIM C(20,20)
20   INPUT N
```

```
60   MAT C = IDN(N,N)
```
(10—58)

Ora la matrice C sarà una matrice d'identità di ordine N.

10-4-4. Lo statement di MAT - INV

L'inverso di una matrice quadrata è un'altra matrice dello stesso ordine, che ha la seguente proprietà: il prodotto di una matrice per il suo inverso è la matrice d'identità. Ad esempio, se I è la matrice d'identità e C è una matrice che è l'inverso della matrice A, ed entrambe sono matrici quadrate in ordine N, allora

$$A*C = C*A = I \qquad (10\text{-}59)$$

Se A è una matrice quadrata, per ottenere il suo inverso possiamo usare il seguente statement BASIC:

```
50   MAT C = INV(A)
```
(10-60)

La forma dello statement è: numero di statement, la parola-chiave MAT, il nome della matrice che sarà l'inverso, il segno di uguale, la parola INV, seguita dalla matrice di cui si vuol ricavare l'inverso, chiusa fra parentesi. Chiaramente si suppone che le matrici A e C siano state opportunamente dimensionate.

Non tutte le matrici quadrate hanno la matrice inversa. Esiste un numero, detto *determinante* di una matrice quadrata: se il determinante di una matrice è zero, l'inverso della matrice non esiste. In casi come questo lo statement di MAT-INV può dare risultati errati. Il determinante di una matrice è generato dalla funzione di libreria BASIC DET, che può essere usata *solo* associata allo statement di MAT-INV. Ad esempio,

```
.
40   MAT C = INV(X)
50   LET G = DET
```

(10-61)

Qui il valore assegnato a G sarà un numero uguale al determinante della matrice X: se G è diverso da zero, C sarà l'inverso della matrice X; se G è uguale a zero, l'inverso della matrice X non esiste.

Si tenga presente che il determinante di una matrice è un numero che ha una sua utilità intrinseca, quindi la funzione di libreria DET non si usa solo a scopo di verifica, cioè per vedere se l'inverso della matrice esiste. Una trattazione sull'uso dei determinanti non rientra negli scopi di un libro sul linguaggio BASIC, ma chi di voi avesse familiarità con questi concetti potrà utilmente servirsi della funzione di libreria DET.

Si tenga presente che la funzione di libreria DET si può usare solo a patto che si sia usata in precedenza la funzione MAT-INV: infatti DET sarà uguale al determinante dell'ultima matrice invertita. Ad esempio,

```
      .
50    MAT C = INV(X)
      .
90    LET G = DET
      .
150   MAT E = INV(H)
160   LET M = DET
```

(10-62)

Ora G sarà il determinante della matrice X. (Si suppone che fra gli statements 50 e 90 non compaia alcuno statement di MAT-INV.) Inoltre M sarà il determinante della matrice H.

10-4-5. Lo statement di MAT - TRN

Si ottiene la trasposizione di una matrice scambiandone le righe con le colonne. Consideriamo ad esempio la matrice

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} \qquad (10\text{-}63)$$

La matrice trasposta è data da

$$\begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix} \qquad (10\text{-}64)$$

Abbiamo in BASIC uno statement che opera la trasposizione di una matrice. Consideriamo ad esempio

```
20   MAT D = TRN(A)                                        (10-65)
```

La matrice D sarà la matrice trasposta della matrice A. Come regola generale, si ha:

$$D(I,J) = A(J,I) \qquad (10\text{-}66)$$

dove D(I,J) ed A(I,J) sono gli elementi delle matrici D e A rispettivamente.

Si noti che la forma dello statement di MAT-TRN è la stessa dello statement di MAT-INV, a parte la presenza della parola TRN al posto della parola INV. Tutte le matrici, anche quelle che non hanno l'inverso, hanno la loro matrice trasposta.

10-4-6. Lo statement di MAT-SIZ

Si è visto come gli statements di MAT-ZER, MAT-CON, etc. possano essere usati per specificare la grandezza di una matrice; comunque questi statements specificano anche i valori dell'elemento. Alcuni compilatori BASIC ammettono l'uso di uno statement capace di specificare la grandezza di una matrice senza specificare gli altri valori. Questo statement ha la forma

```
20   MAT SIZ A(M,N)                                        (10-67)
```

Ora negli statements successivi A sarà trattata come una matrice di M per N elementi. Si tenga presente che, anche se la vostra versione di BASIC non ha lo statement di MAT-SIZ, o un altro equivalente, è possibile ugualmente specificare senza difficoltà la grandezza di una matrice: si può usare ad esempio lo statement di MAT-ZER per specificare la grandezza prima dell'introduzione dei dati per la matrice. All'introduzione dei dati, questi si sovrapporranno ai valori azzerati.

ESERCIZI

10-1. Scrivete uno statement che legga dei dati per scriverli in un vettore A formato da sette elementi. Verificate le fasi operative dello statement scrivendo un opportuno programma in BASIC ed eseguendolo su di un calcolatore.

10-2. Determinate i valori assegnati agli elementi dei vettori A e B nel seguente segmento di programma:

```
 10  DIM A(100),B(100)
 20  MAT READ A(10)
 30  MAT READ B(8)



       .
 90  DATA 1,2,3,4,5,6,7,8
100  DATA 9,10,11,12,13,15,16,17,18,19,20
999  END
```

Verificate i risultati ottenuti scrivendo un programma opportuno ed eseguendolo su di un calcolatore.

10-3. Come si può modificare il programma dell'Esercizio 10-2 in modo che le dimensioni delle matrici A e B possano essere variate con dati d'ingresso?

10-4. Ripetete l'Esercizio 10-2, ma sostituendo agli statements 20 e 30 l'unico statement

```
20 MAT READ A(10),B(8)
```

10-5. Determinate i valori assegnati agli elementi di A e B nel seguente segmento di programma:

```
 10  DIM A(9,2),B(8,3)
 20  MAT READ A
 30  MAT READ B



       .
 90  DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14
100  DATA 15,16,17,18,19,20,21,22,23,24,25
110  DATA 26,27,28,29,30,31,32,33,34,35,36,37
120  DATA 38,39,40,41,42,43,44,45
999  END
```

Verificate i risultati ottenuti scrivendo un opportuno programma in BASIC ed eseguendolo su di un calcolatore.

10-6. Ripetete l'Esercizio 10-5, ma sostituendo gli statements 20 e 30 con lo statement

```
20  MAT READ A,B
```

10-7. Ripetete l'Esercizio 10-5, ma con i primi tre statements sostituiti da

```
10  DIM A(20,20),B(15,18)
20  LET M = 5
30  LET N = 2
40  MAT READ A(M,N),B(3,6)
```

10-8. Illustrate gli effetti degli statements seguenti:

```
10  DIM A(100)
20  MAT INPUT A
```

Si suppone che vengano introdotti per A 15 numeri separati da virgole.

10-9. Scrivete una serie di statements che eseguano le seguenti operazioni: dimensionare un vettore A a 30 righe e 30 colonne; introdurre i dati per il numero effettivo di righe e di colonne; introdurre infine i valori degli elementi. Tutti i dati devono essere introdotti da telescrivente. Quali saranno i valori degli elementi non specificati del vettore? Verificate i risultati ottenuti scrivendo un opportuno programma in BASIC ed eseguendolo su di un calcolatore.

10-10. Illustrate gli effetti dello statement seguente:

```
90  MAT PRINT B,C
```

10-11. Al programma dell'Esercizio 10-2 siano aggiunti i seguenti statements:

```
70  MAT PRINT A
80  MAT PRINT B
```

Commentate l'uscita che verrà stampata. Verificate i risultati ottenuti eseguendo un opportuno programma in BASIC su di un calcolatore.

10-12. Ripetete l'Esercizio 10-11, ma questa volta gli statements aggiunti siano:

```
70  MAT PRINT A,
80  MAT PRINT B,
```

10-13. Ripetete l'Esercizio 10-11, ma questa volta gli statements aggiunti siano:

```
70  MAT PRINT A;
80  MAT PRINT B;
```

10-14. Ripetete l'Esercizio 10-11, ma questa volta aggiungete un solo statements, cioè

```
70 MAT PRINT A;B;
```

10-15. Al programma dell'Esercizio 10-5 siano aggiunti i seguenti statements:

```
70 MAT PRINT A
80 MAT PRINT B
```

Commentate l'uscita che verrà stampata. Verificate i risultati ottenuti eseguendo un opportuno programma in BASIC su di un calcolatore.

10-16. Ripetete l'Esercizo 10-15, ma questa volta gli statements aggiunti siano:

```
70 MAT PRINT A;
80 MAT PRINT B;
```

10-17. Ripetete l'Esercizio 10—15, ma questa volta aggiungete un solo statement, cioè

```
70 MAT PRINT A,B,
```

10-18. Scrivete un programma in BASIC che introduca dei dati in due matrici di grandezza uguale e le sommi. In uscita dovranno risultare i dati appropriati. Verificate il programma eseguendolo su di un calcolatore.

10-19. Scrivete un programma in BASIC che introduca dei dati in quattro matrici A, B, C e D, ciascuna formata da N righe ed M colonne (N ed M devono essere introdotti come dati.) Dev'essere infine stampata la matrice data da A+B—C—D. Verificate il programma eseguendolo su di un calcolatore.

10-20. Ripetete l'Esercizio 10-19, ma questa volta calcolando A+(2)*B—(N+M)*C—D.

10-21. Scrivete un programma in BASIC che introduca dei dati in due matrici A e B. La matrice A è formata da M righe e K colonne, la matrice B da K righe e W colonne. I valori di K, M e W devono essere introdotti come dati. Calcolate la matrice

T = A*B

La matrice T sarà stampata in uscita. Verificate il programma eseguendolo su di un calcolatore.

10-22. Scrivete un programma in BASIC che introduca dei dati in tre matrici quadrate A, B e C, ciascuna formata da M righe e colonne. Il programma dovrà poi calcolare e stampare

A*B*C

Verificate i risultati ottenuti eseguendo il programma su di un calcolatore.

10-23. Ripetete l'Esercizio 10-22, ma calcolando questa volta

C∗B∗A

Confrontate i risultati degli Esercizi 10-22 e 10-23 con diverse matrici 3 × 3. Tenete presente che, come regola generale, è:

C∗B∗A ≠ A∗B∗C

10-24. Modificate il programma dell'Esercizio 10-21 in modo da trovare l'elemento più grande della matrice T.

10-25. Modificate il programma (6-12) in modo che utilizzi i metodi delle matrici.

10-26. Modificate il programma dell'Esercizio 6-15 in modo che utilizzi i metodi delle matrici.

10-27. Vogliamo stampare, con uno statement di MAT PRINT, solo le prime quattro righe e colonne di una matrice che è stata dimensionata a 10 righe e 10 colonne. Per far questo, servitevi dello statement di MAT-ZER, unito ad altri. I dati vanno introdotti da telescrivente.

10-28. Illustrate gli effetti dello statement di MAT-CON. Scrivete un programma in BASIC che mostri l'uso di questo statement.

10-29. Scrivete un programma che introduca i dati per una matrice quadrata di ordine N e ne ricavi l'inverso (N è minore di 30). Il programma deve verificare se l'inverso esiste. Verificate il programma eseguendolo su di un calcolatore.

10-30. Se avete pratica delle soluzioni dei sistemi di equazioni con matrici, scrivete un programma in BASIC che utilizzi i metodi matriciali per risolvere le equazioni dell'Esercizio 6-17.

10-31. Ripetete l'Esercizio 10-30 con le equazioni dell'Esercizio 6-18.

10-32. Illustrate gli effetti dello statement di MAT-TRN. Scrivete ed eseguite un programma in BASIC che evidenzi la vostra esposizione.

CAPITOLO 11

GLI ARCHIVI DI DATI

Nel paragrafo 1-4 abbiamo parlato dell'introduzione e della memorizzazione di un programma in BASIC nella zona del calcolatore ad esso riservata. Un programma così memorizzato è detto *archivio* (*file* in inglese). Gli archivi si usano per memorizzare non solo programmi, ma anche, ad esempio, i dati d'ingresso e di uscita di un programma: parliamo allora di *archivi di dati.*

Questi sono in grado di fornire i dati per i programmi, e possono anche venir stampati sulla telescrivente. L'uso degli archivi di dati può essere molto vantaggioso: supponiamo ad esempio che due programmi distinti utilizzino gli stessi dati. Normalmente tutti i dati dovrebbero essere introdotti due volte (una per programma); se invece collochiamo i dati in un archivio, potranno essere utilizzati da due o più programmi, senza la necessità d'introdurli ogni volta che si esegue un programma.

Ci sono anche altri modi di usare gli archivi di dati. Supponiamo che vengano introdotti i voti riportati da una scolaresca in tre esercitazioni, e che venga scritto un programma che calcoli le medie ed assegni a ciascuno studente una lettera-voto. L'informazione risultante da questi calcoli potrà quindi venir memorizzata in un archivio, utilizzabile poi come dati in ingresso per un altro programma che registri i voti degli studenti nel loro curriculum: non usando un archivio, dovremmo ribattere questa nuova informazione per introdurla nel calcolatore.

In questo capitolo parleremo delle operazioni nelle quali s'impiegano gli archivi di dati. Contrariamente a quanto avviene per la maggior parte del linguaggio BASIC, gli statements relativi agli archivi di dati non sono standard; per questa regione non è detto che gli statements che troverete in questo capitolo siano applicabili al vostro calcolatore. Comunque i concetti di base che esporremo sono validi per quasi tutti i calcolatori, mentre il manuale di BASIC che correda il vostro calcolatore vi dirà la forma esatta degli statements da usare.

11-1. GLI ARCHIVI SEQUENZIALI: CREAZIONE E LETTURA

Un archivio sequenziale è costituito da una sequenza di linee, ciascuna delle quali è identificata da un numero. Così, ad esempio, un programma in BASIC è un archivio siffatto. In particolare, un archivio di dati sequenziale è formato da linee di dati: i dati possono essere stringhe, informazioni numeriche, o le une e le altre insieme. Comunque ogni linea deve iniziare con un numero di linea. Vedremo poi che un archivio siffatto può essere il risultato dell'esecuzione di un programma in BASIC, ma lo si può anche formare da telescrivente. Vediamo come si procede in quest'ultimo caso. Supponiamo di voler introdurre dei dati consistenti nel nome di uno studente, il numero ID ed i voti riportati in tre esercitazioni. Occorre innanzitutto attivare la modalità BASIC (v. (1-18)); in risposta alla domanda OLD OR NEW, si batte NEW. Un tipico ingresso sarà ad esempio questo:

```
OLD OR NEW --- NEW
NEW FILE NAME --- VOTI
>10 "ROSSI" 101 100 90 80
>20 "BIANCHI" 102 80 60 70
>30 "GALLI" 103 70 100 90
>50 "MARINI" 105 80 60 80
>40 "BELLI" 120 100 100 100
>60 "NEGRI" 145 100 50 90
>SAVE                                  (11-1)
```

L'archivio verrà introdotto e memorizzato: ricordate che viene memorizzato nell'area di memorizzazione BASIC a ciò riservata, e che da questo momento potete richiederlo, stamparlo, etc. Si noti che la linea 50 è stata battuta prima della linea 40; allora, una volta memorizzato, l'archivio si disporrà secondo l'esatto ordine numerico, cioè la linea 40 sarà memorizzata prima della linea 50. Abbiamo introdotto le stringhe fra virgolette, il che non sempre è necessario. Abbiamo inoltre separato le voci dei dati con spazi bianchi, invece che con virgole, e questo è un punto che varia da un calcolatore all'altro.

L'archivio può essere manipolato allo stesso modo di un programma in BASIC: ad esempio, se lo richiedete battendo OLD e quindi il suo nome in risposta ad OLD FILE NAME, si può sostituire una linea battendola di nuovo, così come si possono aggiungere nuove linee, etc.

11-1-1. Lettura di un archivio sequenziale

Vediamo ora come si leggono i dati di un archivio durante l'esecuzione di un programma. Si dice che i dati vengono trasmessi fra un programma ed un archivio attraverso un *canale-dati*, spesso definito semplicemente *canale*. A ciascun archivio

utilizzato si assegna un distinto canale-dati, e per far riferimento ad un canale-dati ci si serve di un numero.

Lo statement che assegna a ciascun archivio un canale-dati è lo statement di FILES, che ha la forma

```
10  FILES VOTI,MED                                                    (11-2)
```

vale a dire numero di statement, la parola-chiave FILES, seguita da una lista di nomi di archivi, separati da virgole. Dopo la parola FILES non c'è virgola. Al primo archivio della lista è assegnato il canale-dati 1, al secondo archivio il canale-dati 2, e così via. Nel seguito del programma l'archivio verrà citato non con il suo nome, ma con il suo *numero di canale.*

Lo statement che ordina di fatto al calcolatore di leggere l'informazione da un archivio è lo statement di INPUT: ogni statement di INPUT legge una linea di un archivio. Il seguente è ad esempio uno statement di INPUT che legge un'informazione dall'archivio VOTI (11-1):

```
20  INPUT # 1,N,A$,M,G1,G2,G3                                         (11-3)
```

La forma dello statement di INPUT è: numero di statement, la parola-chiave INPUT, il simbolo # ed un numero che corrisponde al numero del canale, una virgola, una lista di variabili. Le variabili corrispondono all'informazione letta dalla linea di dati.

La seguente è una tipica linea di dati dell'archivio VOTI:

```
10  "ROSSI" 101 100 90 80                                             (11-4)
```

In risposta allo statement (11-3) saranno operate le seguenti assegnazioni:

```
N  =  10
A$ =  "ROSSI"
M  =  101
G1 =  100
G2 =  90
G3 =  80                                                              (11-5)
```

Si tenga presente che dobbiamo assegnare i valori di tutti i dati alle variabili, senza saltare alcun dato. Supponiamo di voler utilizzare solo G1, G2 e G3: anche in questo caso dovremo elencare le variabili corrispondenti ad N, A$ ed M nello state-

ment di INPUT, anche se non le useremo mai. Si tenga presente che le singole voci di una linea vengono lette una per volta; quindi, se vogliamo leggere la quarta, la quinta e la sesta voce, dobbiamo leggere anche la prima, la seconda e la terza. Si noti che il numero di linea è considerato un dato, e va letto anche se non è usato. Se si deve leggere una successiva linea di dati, si dovrà leggere tutta l'informazione contenuta nella linea precedente, anche se non viene usata.

Vediamo ora un nuovo statement; scriveremo poi un programma completo che utilizza i dati provenienti da un archivio sequenziale. Può darsi che non non sappiamo quante linee ci siano in un archivio, e che c'interessi sapere quando la lettura di un archivio è esaurita. Per determinare se siano alla fine di un archivio sequenziale useremo un particolare statement di IF-THEN, della forma

```
50  IF END # 1 THEN 200                                    (11-6)
```

La forma di questo statement è: numero di statement, le parole-chiave IF END, il simbolo # con il numero del canale, la parola THEN e infine un numero, che dev'essere il numero di uno statement che compare nel programma in BASIC. Si tenga presente che alcune versioni di BASIC richiedono la virgola dopo il numero del canale, mentre altre non la richiedono. Le modalità operative di questo statement sono sostanzialmente le stesse di quelle dello statement di IF-THEN di cui si è già parlato: all'esecuzione dello statement 50, se sono stati letti tutti i dati dell'archivio assegnato al canale numero 1, il primo statement ad essere eseguito sarà lo statement 200; se invece restano ancora da leggere dei dati, il successivo statement ad essere eseguito sarà lo statement che vien dopo lo statement numero 50.

Vediamo ora come si applicano i concetti appena esposti ad un programma che legge le informazioni da VOTI, calcola la media di ciascuno studente e infine stampa sulla telescrivente il nome dello studente, il numero ID e la media relativa.

```
10  REM PROGRAMMA CHE LEGGE
15  REM UN ARCHIVIO DATI SEQUENZIALE
20  FILES VOTI
30  PRINT "NOME","NUMERO ID","MEDIA"
40  INPUT # 1,N,A$,M,G1,G2,G3
50  LET B = (G1+G2+G3)/3
60  PRINT A1,M,B
70  IF END # 1 THEN 99
80  GO TO 40
99  END                                                    (11-7)
```

Esaminiamo le fasi operative del programma. Lo statement 20 è lo statement di FILES, che contiene un solo nome di archivio, VOTI, a cui verrà assegnato il canale-dati numero 1. In base allo statement 30 sulla telescrivente viene stampata un'inte-

stazione. All'esecuzione dello statement 40, viene letta la prima linea dell'archivio assegnato al canale 1: si ricordi che, ogni volta che vien eseguito lo statement di INPUT, viene letta una nuova linea dall'archivio. Nello statement 50 si calcola la media, poi, come risultato dello statement 60, sono stampati il nome dello studente, il numero ID e la media. Si noti che il numero di linea N è letto dall'archivio anche se non è utilizzato. Se si è letta l'ultima linea dell'archivio, lo statement 70 passa il controllo allo statement 99, e l'esecuzione ha termine; se invece nell'archivio ci sono altri dati, il controllo torna allo statement 40 e la procedura si ripete.

Si dà per scontato che, quando il programma è eseguito, c'è un archivio denominato VOTI che è stato creato e salvato nella vostra zona di BASIC; seguendo poi la procedura già illustrata in questo paragrafo, l'archivio sarà in seguito memorizzato correttamente.

11-1-2. Denominazione degli archivi

Il nome di un archivio può consistere in una qualunque stringa di non più di sei caratteri. Il primo carattere *deve* essere una lettera.

11-2. GLI ARCHIVI SEQUENZIALI: SCRITTURA

Vediamo ora come si scrivano i dati in un archivio sequenziale durante l'esecuzione di un programma in BASIC, cioè come, in alternativa o in aggiunta alla stampa su telescrivente, si possano scrivere i dati sotto forma di archivio sequenziale, memorizzato nella zona di memorizzazione BASIC. Ma prima di addentrarci in questo discorso, dobbiamo esaminare qualche altro statement.

Lo statement che fa sì che una linea di dati sia scritta in un archivio è PRINT, la cui forma è

```
90   PRINT # 2,N;A$;G1;G2;G3                                    (11-8)
```

cioè numero di statement, la parola-chiave PRINT, il simbolo # seguito dal numero del canale, la virgola, e infine una lista di variabili separate da virgole o da punti e virgola. La lista di variabili corrisponde ai dati da inserire nell'archivio, quali che siano. *La prima voce dei dati dev'essere il numero di linea*: occorre porre attenzione a questo punto perché, se si hanno due linee con lo stesso numero, la seconda sostituirà la prima. Ciò significa che non possono esserci due linee che abbiano lo stesso numero di linea: se sono state scritte due linee con lo stesso numero, la prima è cancellata e al suo posto subentra la seconda. Per separare le variabili si può usare sia una virgola che un punto e virgola; generalmente si usa il punto e virgola, per avere una spaziatura più compatta.

Quando scriviamo dei dati in un archivio, *tutte* le informazioni preesistenti devono essere eliminate: questo si ottiene mediante lo statement di SCRATCH, che ha la forma

```
30  SCRATCH # 2                                                    (11-9)
```

cioè numero di linea, la parola-chiave SCRATCH, il simbolo # ed il numero del canale-dati. State attenti nell'usare questo statement: se sbagliate nello scrivere il numero del canale-dati, sarà cancellato un archivio diverso.

Un altro statement che useremo è lo statement di QUOTE, la cui forma è

```
30  QUOTE # 2                                                      (11-10)
```

cioè numero di linea, la parola-chiave QUOTE, il simbolo ed il numero del canale. Questo statement avverte il calcolatore che le stringhe alfanumeriche vanno messe fra virgolette quando sono scritte nell'archivio corrispondente al canale specificato.

Scriviamo ora un programma che illustri come si scrive un archivio sequenziale. Amplieremo il programma (11-7) in modo che l'uscita non sia scritta solo su telescrivente, ma anche come nuovo archivio, che chiameremo MEDIE. All'archivio MEDIE aggiungeremo una linea a mo' d'intestazione, e cioè

```
5  "CALCOLATORI 101" "PRIMAVERA 1977"                              (11-11)
```

A questa linea daremo il numero 5, sapendo che questo numero è minore degli altri numeri di linea che useremo. Il programma è il seguente:

```
 10  REM PROGRAMMA CHE LEGGE E SCRIVE
 15  REM ARCHIVI DATI SEQUENZIALI
 20  FILES VOTI,MEDIE
 30  PRINT "NOME","NUMERO ID","MEDIA"
 40  QUOTE # 2
 50  SCRATCH # 2
 60  PRINT # 2,5,"CALCOLATORI 101","PRIMAVERA 1997"
 70  INPUT # 1,N,A$,M,G1,G2,G3
 80  LET B = (G1+G2+G3)/3
 90  PRINT A$,M,B
100  PRINT # 2,N,A$,M,B
110  IF END # 1 THEN 999
120  GO TO 70
999  END                                                           (11-12)
```

Vediamone le fasi operative. Nello statement 20 indichiamo che VOTI sarà trasmesso sul canale-dati 1, e MEDIE sarà trasmesso sul canale-dati 2. Lo statement 30 stampa un'intestazione sulla telescrivente. Lo statement 40 dice che le stringhe saranno messe fra virgolette, quando saranno registrate sul canale 2. Lo statement 50 cancella tutte le informazioni preesistenti sull'archivio corrispondente al canale-dati 2, e fa partire la memorizzazione dell'informazione dall'inizio dell'archivio. Si tenga presente che, se VOTI non è mai stato utilizzato prima, cioè se non esiste già un archivio di questo nome, sarà l'esecuzione del programma a crearlo; se esiste già, verrà cancellato. E veniamo all'esecuzione dello statement 60, che scrive:

```
5 "CALCOLATORI 101" PRIMAVERA 1977"
```

nella prima linea dell'archivio MEDIE.

Lo statement 70 fa sì che venga letta la prima linea dell'archivio collegato al canale 1 (VOTI). All'esecuzione dello statement 80 si calcola la media, mentre lo statement 90 fa stampare i dati sulla telescrivente.

Lo statement 100 fa sì che il numero di linea N, il nome dello studente A$, il numero ID M e la media B vengano scritti nell'archivio MEDIE. Il programma continua nei suoi cicli di esecuzione finché tutti i dati saranno letti dall'archivio collegato al canale numero 1 (VOTI) e sarà scritta l'uscita opportuna.

Si noti che, ogni volta che è eseguito un nuovo statement di PRINT, viene scritta la linea immediatamente successiva nell'archivio in questione; analogamente, ogni volta che è eseguito un nuovo statement di INPUT, viene letta la successiva espressione dall'archivio in questione. Quindi dobbiamo procedere lungo l'archivio una linea alla volta: se vogliamo leggere una sola linea, posta verso la fine dell'archivio stesso, dovremo passare attraverso tutte le linee precedenti. Se ad esempio vogliamo leggere soltanto la quindicesima linea, dovremo eseguire 15 statements di INPUT.

11-2-1. Ridenominazione degli archivi

Spesso si ha bisogno di ridenominare gli archivi. Ad esempio, il programma (11-12) creava un archivio detto MEDIE; supponiamo ora di volerlo usare come dati d'ingresso per un altro programma, e che l'archivio d'ingresso per questo programma si chiami MED: possiamo dare a MEDIE il nuovo nome di MED, usando ad esempio questa sequenza:

```
>OLD
>OLD FILE NAME ---- MED
>SCRATCH
>OLD
>OLD FILE NAME ---- MEDIE
>RENAME MED                                  (11-13)
```

Spieghiamo la sequenza: prima richiediamo il vecchio archivio MED e lo cancelliamo. Quest'operazione è necessaria se dobbiamo ridenominare un altro archivio con lo stesso nome: la parola usata è SCRATCH. A questo punto si richiede un altro vecchio archivio, che è ora MEDIE: in risposta al >, battete il comando di sistema BASIC RENAME, seguito dal nuovo nome che intendete dare all'archivio. Così la procedura (11-13) darà all'archivio MEDIE il nuovo nome di MED: si tenga presente che, se non ci fosse stato un archivio preesistente denominato MED, i primi tre passaggi non sarebbero stati necessari.

11-3. GLI ARCHIVI CASUALI: SCRITTURA

Un archivio sequenziale dev'essere letto nell'ordine; cioè la prima linea si legge prima della seconda, e così via. Ma questo può rivelarsi un procedimento lento: volendo ad esempio un'informazione contenuta in una sola linea, o in poche linee prossime alla fine dell'archivio, siamo costretti (v. par. 11-2) a passare tutto l'archivio. Ci sono degli archivi detti *archivi casuali* (*random files* in inglese) per i quali questo modo di procedere non si pone; così, ad esempio, si possono leggere i dati posti alla fine dell'archivio senza dover leggere i dati precedenti. Questi archivi non sono disposti secondo un formato di linee successive, ma ciascuna *voce* dei dati è memorizzata in una distinta locazione. Queste locazioni sono numerate progressivamente, mentre un puntatore indica una locazione dei dati: questa locazione sarà quella che verrà letta o scritta. Vedremo che esistono dei comandi che muovono il puntatore ad una determinata locazione dell'archivio: in tal modo possiamo leggere o scrivere senza dover percorrere l'intero archivio, potendo muovere il puntatore a piacere avanti e indietro. Ci occuperemo ora di come avvengono la scrittura e la lettura da un archivio casuale, e poi dei comandi concernenti il puntatore. Se non si ricorre a questi comandi, il puntatore procede da una voce a quella successiva: ad esempio, se l'ultimo dato letto dall'archivio è quello situato nella terza locazione di memoria, in assenza di comandi di puntatore il successivo termine ad essere letto sarà quello posto nella quarta locazione di memoria.

Gli archivi casuali possono contenere o numeri o stringhe, *ma non gli uni e le altre insieme*. I contenuti dell'archivio saranno specificati nello statement di FILES. Come negli archivi sequenziali, a ciascun archivio viene assegnato un canale-dati per mezzo dello statement di FILES. Supponiamo di avere i due archivi ALF e NUM, di cui l'archivio ALF è formato da stringhe, mentre l'archivio NUM da numeri. Lo statement FILES relativo avrà ad esempio la forma

```
10   FILES ALF$ 15, NUM%                                        (11-14)
```

Si noti che i nomi degli archivi sono ALF e NUM. Nello statement di FILES inseriamo in aggiunta dei suffissi che indicano il tipo di archivio: il segno % designa i dati

numerici; il segno $ seguito da un numero designa le stringhe, mentre il numero specifica il numero massimo di caratteri di ciascuna stringa. Quindi lo statement (11-14) alloca il canale 1 ad ALF, che è formato da variabili-stringa, ciascuna della lunghezza massima di 15 caratteri; inoltre alloca il canale 2 all'archivio NUM, che è formato da dati numerici.

Quando si crea un archivio casuale, non si può in maniera semplice introdurre dei dati da telescrivente, come si fa con gli archivi sequenziali: per introdurre dei dati, occorre scrivere un programma in BASIC apposito. Lo statement usato è lo statement di WRITE: ecco un esempio tipico:

```
40   WRITE:1,A,B,(C+B)*A                                   (11-15)
```

La forma di questo statement è: numero di statement, la parola-chiave WRITE, due punti, un numero, che è il numero del canale-dati, una virgola, ed infine una lista di variabili o di espressioni, separate da virgole. Si noti la presenza dei due punti al posto del simbolo #: i due punti indicano il canale-dati nel caso di archivi casuali.

Vediamo ora un programma che scrive un archivio casuale:

```
10   REM PROGRAMMA DI SCRITTURA
15   REM DI UN ARCHIVIO CASUALE
20   FILES INDICI%
30   INPUT A,B,C,D
40   IF A < 0 THEN 99
50   WRITE:1,A,B,C,D
60   GO TO 30
99   END                                                   (11-16)
```

Nello statement 20 un archivio numerico casuale, di nome INDICI, viene assegnato al canale 1. Quindi vengono introdotti quattro numeri che, se A non è negativo, vengono scritti sul canale-dati 1. Si noti che, all'esecuzione del programma, il puntatore è posizionato automaticamente sulla prima locazione dei dati, in cui sarà quindi scritto il valore introdotto per A. Allo stesso modo il valore introdotto per B sarà memorizzato nella locazione 2, e così via. Il controllo torna poi allo statement 30, e vengono introdotti nuovi valori per A,B,C e D: ora il valore assegnato ad A sarà memorizzato nella locazione 5, e così via. Quando è introdotto per A un numero negativo, l'esecuzione termina.

Con molti calcolatori non è possibile eseguire questo programma così com'è, ma bisogna costituire prima un archivio denominato INDICI. (Si tenga presente che, se si ha a che fare con un archivio sequenziale, è il programma stesso a formarlo.) A tal fine dovremo battere i seguenti comandi di sistema BASIC:

```
>NEW
>NEW FILE NAME --- INDICI
>SAVE                                                      (11-17)
```

Così facendo, creiamo un nuovo archivio che conserviamo con SAVE. *Non si battono altri dati.* Una volta creato l'archivio, non dobbiamo più ripetere la procedura (11-17) quando il programma (11-16) è rieseguito. Si tenga presente che con alcuni calcolatori non è necessario mettere in atto la procedura (11-11). Consultate il manuale di BASIC annesso al calcolatore utilizzato per vedere se è necessaria e se non impiegandola l'archivio sarà conservato.

Supponiamo ora di aver creato l'archivio e di eseguire il programma (11-16); supponiamo inoltre che, in risposta al punto interrogativo di avviso, vengano battuti i seguenti dati:

```
? 176,90,70,100
? 185,80,100,80
? 181,90,90,100
? -1,0,0,0                                        (11-18)
```

La tabella seguente riporta le locazioni dei dati per ciascun elemento dell'archivio denominato INDICI:

Locazioni dei dati	**Valori memorizzati**
1	176
2	90
3	70
4	100
5	185
6	80
7	100
8	80
9	181
10	90
11	90
12	100

Supponiamo ora di tornare ad eseguire il programma (11-16) e d'introdurre i dati a partire dalla locazione 1: quando si scrive un nuovo valore, si cancella il vecchio valore. Allora, *rieseguendo* il programma (11-16) con i dati

```
? 192,100,100,100
? 253,40,40,40
? -1,0,0,0                                        (11-19)
```

i valori memorizzati e le rispettive locazioni dei dati saranno i seguenti:

Locazioni dei dati	Valori memorizzati
1	192
2	100
3	100
4	100
5	253
6	40
7	40
8	40
9	181
10	90
11	90
12	100

Vale a dire che i nuovi otto valori letti saranno scritti al posto dei primi otto valori dell'archivio; inoltre, dato che non vengono letti altri dati, i dati memorizzati nelle locazioni 9,10,11 e 12 restano invariati.

Abbiamo visto i principi essenziali della scrittura di un archivio casuale; nel prossimo paragrafo vedremo come questo archivio viene usato.

11-4. LETTURA DI UN ARCHIVIO CASUALE

Esaminiamo ora le procedure essenziali che si usano per leggere gli archivi casuali. La lettura si ottiene con uno statement di READ della forma

```
30   READ:1,A,B,C,D                                    (11-20)
```

cioè numero di statement, la parola-chiave READ, due punti, un numero, che è il numero del canale, seguito da una lista di variabili. Le variabili vengono lette nell'ordine e, quando parte l'esecuzione del programma, il puntatore è sulla prima locazione dei dati. Ad ogni valore letto, il puntatore avanza di una locazione. Quindi, se lo statement 30 è il primo statement di READ che compare nel programma per il canale 1, A sarà letta dalla locazione 1, B dalla locazione 2, e così via. Se lo statement 30 viene eseguito di nuovo, A sarà letta dalla locazione 5, B dalla locazione 6, e così via.

Prima di andare avanti, esamineremo due funzioni BASIC di libreria: la prima ci permette di determinare la posizione attuale del puntatore, allo scopo di determinare la prossima locazione che verrà letta o scritta; la seconda fornisce la locazione dell'ultimo elemento dell'archivio.

11-4-1. La funzione di libreria LOC

Con la funzione di libreria LOC si ricava la posizione attuale del puntatore. Ad esempio,

```
20   LET B = LOC(1)                                        (11-21)
```

All'esecuzione di questo statement B sarà uguale alla posizione attuale del puntatore per quanto riguarda l'archivio assegnato al canale-dati 1. Se la prossima locazione da leggere o scrivere è la terza, B = 3. Se vogliamo conoscere la posizione attuale del puntatore per l'archivio assegnato al canale - dati 2, scriveremo:

```
30   LET C = LOC(2)                                        (11-22)
```

11-4-2. La funzione di libreria LOF

Con la funzione di libreria LOF si può conoscere la locazione dell'ultimo elemento dell'archivio. Ad esempio,

```
30   LET F = LOF(1)                                        (11-23)
```

All'esecuzione dello statement 30, F sarà uguale alla locazione dell'ultimo elemento dell'archivio assegnato al canale 1. Se ad esempio l'archivio è formato da 32 elementi, F è uguale a 32.

Si noti che il numero fra parentesi, sia nella funzione LOC che nella funzione LOF, si riferisce al canale - dati.

Esaminiamo ora un programma con il quale leggiamo l'archivio casuale INDICI, che si è scritto con il programma (11-16). Supponiamo che a ciascun gruppo di quattro numeri corrispondano il numero ID di uno studente ed i suoi tre voti: vogliamo calcolare la media dei voti, e quindi stampare il numero ID e la media. Questa stessa informazione sarà scritta inoltre in un archivio casuale denominato MED. Il programma è il seguente:

```
10   REM PROGRAMMA CHE LEGGE E SCRIVE
15   REM ARCHIVI CASUALI
20   FILES INDICI%,MED%
30   PRINT "NUMERO ID","MEDIA"
40   READ:1,N,G1,G2,G3
50   LET A = (G1+G2+G3)/3
60   PRINT N,A
70   WRITE:2,N,A
```

```
80  IF LOC(1) > LOF(1) THEN 99
90  GO TO 40
99  END
```

(11-24)

Vediamone le fasi operative. Nello statement 20 vengono predisposti i canali-dati per due archivi casuali di dati: INDICI è assegnato al canale 1, MED al canale 2. Si assume che per MED sia stata messa in opera la procedura (11-17) e che i dati di INDICI siano così ordinati: il dato posto nella locazione 1 è il numero ID di uno studente, mentre i dati posti nelle locazioni 2, 3 e 4 sono i voti riportati dallo studente in tre esercitazioni. Questa sequenza si ripete sempre uguale. Lo statement 30 fa stampare un'intestazione sulla telescrivente. Quando viene eseguito lo statement 40 (per la prima volta) i primi quattro valori dei dati vengono letti da INDICI ed assegnati rispettivamente ad N, G1, G2 e G3. Nello statement 50 si calcola la media di G1, G2 e G3. Lo statement 60 provvede alla stampa dell'informazione sulla telescrivente. Analogamente, lo statement 70 provvede alla scrittura del numero ID dello studente e della sua media in MED, cioè nell'archivio assegnato al canale 2.

E passiamo allo statement 80. Se il puntatore è andato oltre l'ultima locazione di INDICI, l'elaborazione termina. Se invece LOC(1) non è maggiore di LOF(1), vuol dire che il puntatore di canale non è andato oltre l'ultima locazione dei dati. Viene allora eseguito lo statement 90, il controllo torna allo statement 40, altri dati vengono letti dall'archivio del canale 1 e la procedura ciclica si ripete.

11-4-3. Lo statement di FILE

Quando si esegue un programma come l'(11-24), lo statement di FILES richiede tutti gli archivi necessari, che sono quindi assegnati a dei canali-dati. Capita che uno stesso programma venga eseguito più volte utilizzando archivi con nomi diversi, come ad esempio il programma (11-24) (che calcola la media dei voti) quando lo si voglia eseguire con i voti di classi differenti: i dati di ciascuna classe si potranno memorizzare in un archivio distinto, ciascuno contrassegnato con un nome diverso. Se quindi vogliamo usare nel programma (11-24) dei nomi di archivi differenti, lo statement 20 dovrà essere modificato ad ogni esecuzione del programma.

In realtà disponiamo di uno statement BASIC che permette d'introdurre il nome di un archivio in qualità di dato: si tratta dello statement di FILE. Vediamone l'uso.

```
10  PRINT "INTRODURRE IL NOME",
15  PRINT "DELL'ARCHIVIO"
20  INPUT A$
30  FILE: 1,A$
```

(11-25)

Quando il programma è eseguito, in risposta agli statements 10 e 20 (si ricordi che il 20 attiva l'introduzione e la memorizzazione di una variabile-stringa) apparirà quanto segue:

```
INTRODURRE IL NOME DELL'ARCHIVIO
?
```

In risposta al ? l'utilizzatore batte il nome dell'archivio con il suffisso adatto. Ad esempio,

```
? INDICI%                                            (11-26)
```

che avrà lo stesso effetto che se ai primi tre statements si sostituisse

```
10  FILES INDICI%
```

La forma dello statement di FILE è: numero di statement, la parola-chiave FILE, due punti, un numero che assegna il canale-dati, la virgola, ed infine una variabile-stringa. A questa variabile-stringa dev'essere assegnato un valore prima che lo statement di FILE sia eseguito.

Si può usare uno statement di FILE distinto per ciascun archivio. Consideriamo ad esempio il seguente segmento di programma:

```
10  PRINT "INTRODURRE I NOMI DEGLI ARCHIVI",
15  PRINT "DI INGRESSO E DI USCITA"
20  INPUT A$,B$
30  FILE:1,A$
40  FILE:2,B$                                        (11-27)
```

L'archivio il cui nome è introdotto per primo sarà assegnato al canale-dati 1, quello introdotto per secondo sarà assegnato al canale-dati 2.

Si tenga presente che lo statement di FILE si usa anche con gli archivi sequenziali: la forma è la stessa, solo che, invece dei due punti, compare il simbolo #. Ad esempio, per introdurre il nome di un archivio sequenziale, si userà la forma

```
20  FILE # 1,A$                                      (11-28)
```

11-5. TRATTAMENTO DEGLI ARCHIVI CASUALI

Finora abbiamo usato gli archivi casuali come se fossero archivi sequenziali; infatti li abbiamo letti e scritti nell'ordine. Si ricordi che c'è un puntatore per ciascun archivio (canale-dati): il puntatore indica la prima locazione dei dati da leggere o scrivere. In seguito all'esecuzione di uno statement di READ o di WRITE, il puntatore avanza di una locazione. Quest'operazione è sostanzialmente sequenziale, ma può anche non esserlo. Si può spostare il puntatore mediante lo statement di SET, che ha la forma

```
50   SET:1,A                                        (11-29)
```

cioè numero di statement, la parola-chiave SET, due punti, un numero, uguale al canale-dati desiderato, la virgola, una variabile. Il puntatore sarà posizionato sulla locazione definita dal valore assegnato ad A, ove si assume che il valore assegnato ad A sia un numero intero. Consideriamo ad esempio la sequenza.

```
40   N = 20
50   SET: 1,N
60   READ: 1,A1
70   READ: 1,A2                                     (11-30)
```

Il valore di A1 sarà letto dalla ventesima locazione dell'archivio corrispondente al canale 1, poi sarà letto il valore di A2 dalla ventunesima locazione dell'archivio corrispondente al canale 1.

Per illustrare l'uso dello statement di SET, prendiamo in esame il problema seguente: supponiamo di avere un archivio costituito dai numeri ID di tutti gli studenti di una classe e che questi numeri siano elencati secondo l'ordine numerico, partendo dal numero più basso. Vogliamo vedere se un determinato studente fa parte della classe. Si potrebbe leggere l'intera lista e confrontare i numeri, ma questo ci farebbe perdere del tempo, soprattutto se la lista è lunga. Se abbiamo a che fare con archivi sequenziali, l'uso di una tale procedura antieconomica è inevitabile; possiamo invece disporre di una procedura molto più efficiente se ricorriamo ad archivi casuali ed allo statement di SET. Esaminiamo questo punto. Supponiamo che la lista dei numeri ID sia la seguente:

Locazione	Numero
1	10
2	25
3	27
4	36
5	45
6	49
7	56
8	79
9	83
10	99
11	123
12	126
13	2334
14	4567
15	6789

(11-31)

Supponiamo ora di voler stabilire se il numero 4567 è presente nella lista. Invece di scorrere la lista, possiamo operare in questo modo: consideriamo la linea posta a metà della lista: vi compare il numero 79 e la locazione è 8. Ora, $4567 > 79$, e quindi, se il numero è presente nella lista, comparirà oltre la locazione 8, per cui ci basta esaminare la seconda metà della lista. Andiamo ora alla linea mediana della seconda metà della lista: vi troviamo la locazione 12, ed il valore memorizzato in essa è 126. Ora, $4567 > 126$, per cui, se il numero è presente nella lista, dev'essere in una delle locazioni che vanno dalla 13 alla 15. Dividiamo ancora a metà quel che resta della lista, e proviamo con la locazione 14. Qui il numero è 4567, che è uguale al numero che cercavamo. Si noti che l'abbiamo trovato dopo tre tentativi, invece dei quattordici che sarebbero stati necessari se i dati fossero stati memorizzati in un archivio sequenziale. Quindi la facoltà di spostare il puntatore ci permette di scorrere l'archivio in modo più efficiente. Vediamo ora un programma che realizza questa procedura.

```
10  REM PROCEDURA DI RICERCA
15  REM ENTRO UN ARCHIVIO
20  PRINT "INTRODURRE IL NOME",
25  PRINT "DELL'ARCHIVIO"
30  INPUT A$
40  FILE:1,A$
50  PRINT "INTRODURRE IL NUMERO ID",
55  PRINT "DA CERCARE"
60  INPUT N
70  IF N < 0 THEN 999
80  LET P1 = 1
```

```
 90  LET P3 = LOF (1)
100  IF P3-P1 > = 1 THEN 180
110  SET: 1,P1
120  READ:1,N1
130  IF N1 <> N THEN 160
140  PRINT "ID TROVATO NELLA",
145  PRINT "LOCAZIONE";P1
150  GO TO 50
160  PRINT "ID NON E' NELL'ARCHIVIO"
170  GO TO 50
180  LET P2 = INT((P1+P3)/2)
190  SET:1,P2
200  READ:1,N1
210  IF N1 = N THEN 270
220  IF N1 < N THEN 250
230  LET P3 = P2-1
240  GO TO 100
250  LET P1 = P2+1
260  GO TO 100
270  PRINT "ID TROVATO NELLA",
275  PRINT "LOCAZIONE";P2
280  GO TO 50
999  END                                              (11-32)
```

Analizziamone le fasi operative. In figura 11-1 è rappresentato il diagramma di flusso del programma. Negli statements 20-40 viene introdotto il nome di un archivio, che è assegnato al canale-dati 1. L'esecuzione dello statement 60 assegna ad N il numero ID che ci si propone di cercare. Facciamo in modo che il programma si ripeta, nel senso che, esaminato l'archivio e stampata l'informazione, il calcolatore chiederà un nuovo numero ID, per cui la procedura si ripeterà: se per N viene introdotto un numero negativo, lo statement 70 porrà fine alla procedura. In questo modo l'utilizzatore può interrompere l'operazione.

Consideriamo ora la routine di ricerca. Negli statements 80 e 90 poniamo le variabili P1 e P3 uguali rispettivamente ad 1 e al più alto numero di locazione presente nell'archivio. Nello statement 100 verifichiamo se P1−P3 è maggiore o uguale ad uno. In caso contrario, P1 = P3, e allora c'è un solo numero nell'archivio o, come vedremo, nella parte dell'archivio su cui è effettuata la ricerca. In questo caso poniamo il puntatore uguale a P1 e leggiamo l'archivio (statements 110 e 120). Il valore letto è denominato N1. Se N1 = N, abbiamo trovato il numero che cercavamo; poi lo statement 140 fa stampare quest'informazione e la locazione dell'archivio. Se N1≠ N, lo statement 160 fa stampare la frase ID NON E' NELL'ARCHIVIO. Il controllo ritorna quindi allo statement 50, e si può introdurre un nuovo numero ID. Il procedimento si ripete, e così la ricerca.
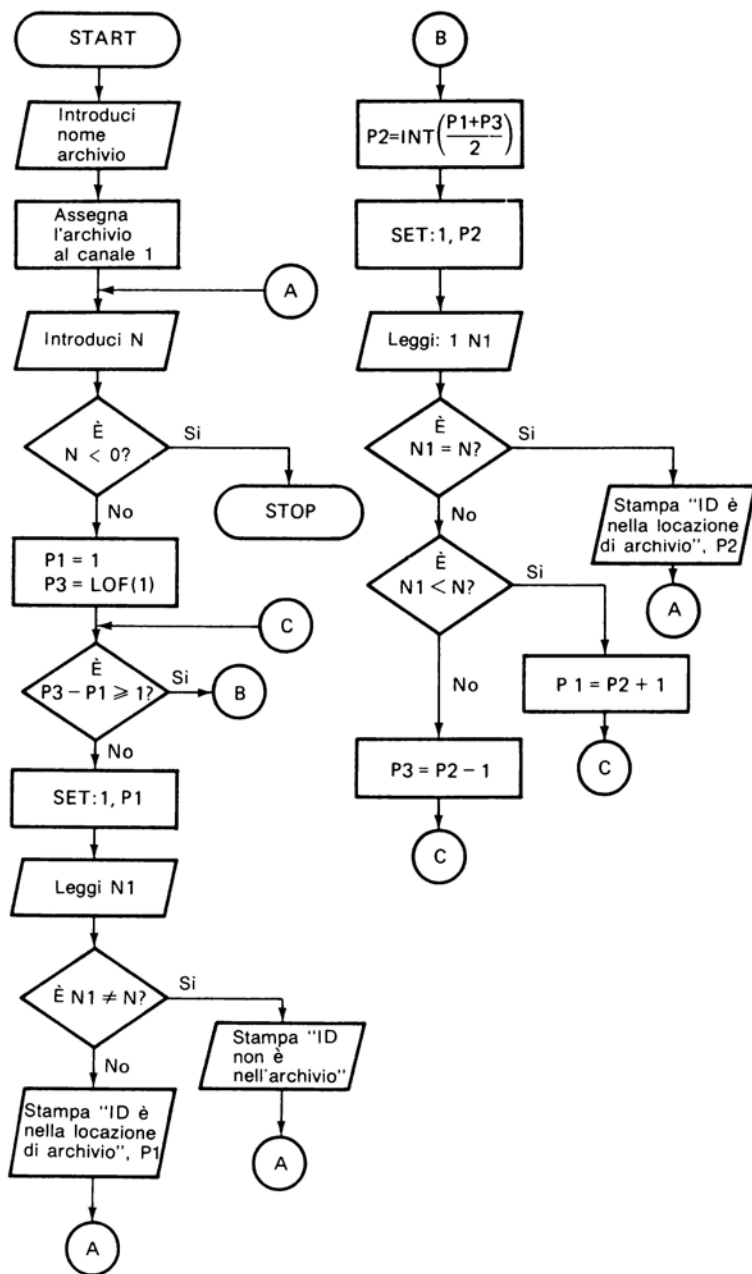
Figura 11.1 — Diagramma di flusso del programma (11-32) che effettua una ricerca all'interno di un archivio casuale.

Generalmente all'inizio della ricerca P3 > P1, dato che l'archivio è formato da molti elementi. Quindi il controllo passa allo statement 180. Vogliamo ora posizionare il puntatore alla metà dell'archivio: la posizione del puntatore sarà dunque P2. Nello statement 180 si calcola

P2 = INT((P1+P3)/2)

P2 sarà il numero intero il cui valore è il più prossimo al numero della locazione corrispondente alla metà dell'archivio. Negli statements 190 e 200 il puntatore è posto a P2, e il valore di questa locazione è letto come N1. Nella linea 210 verifichiamo se N1 = N. In caso affermativo, stampiamo che il numero ID richiesto fa parte dell'archivio, e ne diamo la locazione (statement 270). Il controllo torna quindi allo statement 50, e possiamo dunque ricercare un nuovo numero ID, ripetendo il procedimento.

Supponiamo ora che, all'esecuzione dello statement 210, N1 sia diverso da N (N1 ≠N). Verifichiamo allora se N1 è maggiore o minore di N. Nello statement 220 si verifica se N1 < N: in caso affermativo, il numero ID che cerchiamo è maggiore del valore memorizzato alla metà dell'archivio, e quindi, se il numero ID è presente, dovrà essere memorizzato nella seconda metà dell'archivio, e di conseguenza è questa parte dell'archivio che dobbiamo esaminare. Per far questo, sostituiamo P2+1 a P1 (statement 250): quindi P1, che rappresenta la prima locazione dei dati, da 1 diventa P2+1. Ad esempio, tornando all'archivio (11-31), 9 è considerata come la più bassa locazione dei dati, per cui l'archivio sarà esaminato dalla locazione 9 alla locazione 15. Il controllo torna allo statement 100. Si riprende da capo la procedura di ricerca appena illustrata; infatti, se P3—P1 non è uguale o maggiore di uno, nella metà dell'archivio che stiamo esaminando compare un solo elemento; questo viene confrontato con N, e l'esatta informazione viene stampata sulla telescrivente. Se invece P3—P1 > = 1, il controllo torna allo statement 180: il segmento di archivio in esame viene ancora una volta diviso a metà, e il procedimento si ripete.

Finora abbiamo considerato quello che si verifica quando N1 < N (statement 220). Se è vero il contrario, dobbiamo esaminare la prima metà dell'archivio: lasceremo allora P1 invariato, ma abbasseremo P3 a P2—1. Ora la locazione più alta del segmento di archivio in esame è minore del valore mediano originario: nel caso dell'archivio (11-31), limiteremo il nostro lavoro di verifica alle locazioni 1-7. In altre parole, il controllo torna allo statement 100, e il procedimento si ripete.

Si tenga presente che ad ogni passaggio la parte di archivio su cui si effettua la ricerca ha come limiti le locazioni P1 e P3. L'estensione dell'archivio definita da queste due locazioni viene poi divisa a metà, e quindi possiamo stabilire rapidamente se il numero ID cercato è presente nell'archivio.

Noi abbiamo illustrato la procedura di ricerca con dati numerici; in realtà si possono usare anche stringhe di dati. A questo proposito, si ricordi che alle stringhe di dati si assegnano dei valori numerici in base al loro ordine alfabetico (v. par 8-2).

Pertanto si può modificare il programma (11-32) in modo che possa ricercare dei nomi, piuttosto che dei numeri.

ESERCIZI

11-1. Dite in cosa consiste l'utilità degli archivi.

11-2. Create un archivio sequenziale, ciascun elemento del quale sia costituito dal nome di uno studente e da quattro voti.

11-3. Ciascun venditore di una ditta vende quattro differenti tipi di pezzi. Create un archivio sequenziale, ciascun elemento del quale contenga il nome di un venditore e quattro numeri, uguali al totale delle sue vendite per ciascun pezzo. Introducete 10 elementi diversi dopo aver preparato i dati.

11-4. Spiegate il concetto di canale - dati.

11-5. Scrivete un programma che legga i dati dell'Esercizio 11-3, e quindi calcoli la commissione globale di ciascun venditore. La commissione per ciascuna vendita è:

Numero del pezzo	**Commissione**
1	L. 100
2	L. 400
3	L. 250
4	L. 1,500

Sulla telescrivente dovrà essere stampata una tabella elencante i nomi di tutti i venditori e la relativa commissione globale. Dovranno essere introdotte sempre dieci linee di dati.

11-6. Ripetete l'Esercizio 11-5, ma supponendo d'ignorare la lunghezza dell'archivio in ingresso.

11-7. Modificate il programma dell'Esercizio 11-6 in modo che il risultato in uscita venga scritto in un archivio sequenziale.

11-8. Scrivete un programma in cui l'ingresso di un archivio sequenziale è dato dall'uscita dell'Esercizio 11-7. Il programma dovrà servire a scrivere a ciascun venditore una lettera che cominci così:

EGREGIO SIG. (Nome)
LA SUA COMMISSIONE GLOBALE MENSILE E'
(fornire il valore)

DISTINTI SALUTI
DITTA XYZ

11-9. Perché, quando si scrive un archivio sequenziale, si deve usare lo statement SCRATCH # 2?

11-10. Qual è il significato dello statement QUOTE # 2?

11-11. Spiegate come si dà un nuovo nome ad un archivio.

11-12. Illustrate la differenza fra archivio sequenziale ed archivio casuale.

11-13. Illustrate il significato dello statement

```
10  FILES LIBRO$15,CASA%
```

11-14. Ripetete l'Esercizio 11-3, ma con un archivio casuale. Identificate i venditori non per mezzo del nome, ma per mezzo del numero ID. I venditori sono dieci. Preparate i dati appropriati.

11-15. Ripetete l'Esercizio 11-14, ma questa volta con la possibilità d'introdurre un numero arbitrario di vendite. Cioè fate in modo che il programma sia rieseguito fino a quando l'operatore dia l'opportuno comando di fine esecuzione.

11-16. Elencate le locazioni dei dati per ciascun elemento introdotto nell'archivio dell'Esercizio 11-14.

11-17. Scrivete un programma che legga l'archivio costituito nell'Esercizio 11-15, e calcoli la commissione di ciascun venditore. Il numero ID e la commissione dovranno essere stampati su telescrivente. Sapete che sono stati introdotti dati per dieci venditori. La commissione per ciascun articolo è data nell'Esercizio 11-5. I dati, per ciascun venditore, sono introdotti nell'ordine seguente: numero ID, le vendite del pezzo 1, le vendite del pezzo 2, le vendite del pezzo 3, le vendite del pezzo 4.

11-18. Illustrate il significato degli statements

```
20  LET F = LOC(1)
30  LET G = LOC(2)
```

11-19. Illustrate il significato dello statement

```
20  LET M = LOF(2)
```

11-20. Ripetete l'Esercizio 11-17, ma con un numero arbitrario di venditori. (Si ignori quanti dati vi siano in ingresso nell'archivio.)

11-21. Modificate il programma dell'Esercizio 11-8 in modo che il risultato in uscita venga anche scritto in un altro archivio casuale.

11-22. Ripetete l'Esercizio 11-8, ma per ottenere questa volta, dall'archivio dell'Esercizio 11-21, l'informazione relativa alle commissioni. Avete a disposizione anche altri due archivi casuali; l'uno elenca i nomi dei venditori, l'altro i loro numeri ID. Le locazioni dei dati sono le stesse nei due archivi: ad esempio, il venditore il cui nome si trova nella locazione 3 dell'archivio dei nomi avrà il suo numero ID nella locazione 3 dell'archivio dei numeri ID.

11-23. Illustrate il significato dello statement

```
30 FILE:1,B$
```

Verificate le vostre affermazioni scrivendo un programma opportuno ed eseguendolo su di un calcolatore.

11-24. Ripetete l'Esercizio 11-23 con lo statement

```
30  FILE#1,B$
```

11-25. Illustrate il significato dello statement

```
100  SET:1,B
```

11-26. Il programma (11-32) richiede che i numeri ID vengano memorizzati in ordine numerico. Supponiamo che questo non avvenga. Scrivete un programma il cui ingresso sia un archivio casuale costituito da numeri ID memorizzati in ordine casuale. In uscita dovremo avere un altro archivio casuale in cui dovranno comparire tutti i numeri dell'archivio, ma disposti nell'ordine numerico. I numeri siano interi compresi fra 1 e 9999.

11-27. Modificate il programma (11-32) in modo che cerchi un nome che v'interessa in una lista di nomi alfabetici. Dovrà essere stampato un messaggio indicante se il nome è presente o no nella lista. Se è presente, dovrà essere pure fornita la sua locazione di memoria.

APPENDICE A

GLOSSARIO DEI TERMINI BASIC

Diamo qui un elenco dei vari termini, espressioni ed operazioni usati nel linguaggio di programmazione BASIC. Accanto ad ogni voce troverete un esempio di utilizzo ed il numero del paragrafo in cui è stata trattata.

Termine	Esempio		Par.
Addizione +	40	LET X = Y+Z	2-1
Asterisco *	50	LET B = A*C	2-1
CHANGE	20	CHANGE A$ TO B	8-3
CHANGE	30	CHANGE B TO A$	8-3
DATA	90	DATA 100,LIBRO,30	3-1
DEF	20	DEF FNA(X,Y) = X↑2+Y↑2	7-1
DIM	10	DIM A(30),B(20,30)	6-2
Diverso da < >	10	IF A < > B THEN 20	4-2
Divisione /	50	LET A = X/Y	2-1
Elevamento a potenza ↑	80	LET A = (B+C)↑D	2-2
END	99	END	1-2
FILE	20	FILE:1,A$	11-4
FILES	30	FILES VOTI,MEDIE	11-1
FNEND	150	FNEND	7-2
FOR-TO	20	FOR K = 1 TO N STEP N1	5-1
GOSUB	50	GOSUB 300	7-4
GO TO	90	GO TO 10	4-1
IF END-THEN	50	IF END # 2 THEN 150	11-1
IF-GO TO	30	IF A = B GO TO 100	4-2
IF-THEN	90	IF A < > B THEN 500	4-2
INPUT	10	INPUT A,B$,C	3-3
INPUT	20	INPUT #1,N,A$,B,C,D	11-1
LET	30	LET A+B	2-5

Termine	Esempio		Par.
Maggiore di >	20	IF A > B+1 THEN 250	4-2
Maggiore o uguale a >=	50	IF A >= 150 THEN 200	4-2
MAT-CON	20	MAT B = CON(4,3)	10-4
MAT-IDN	20	MAT B = IDN(N,N)	10-4
MAT-INPUT	30	MAT INPUT Y	10-1
MAT-INV	90	MAT B = INV(Z)	10-4
MAT-meno –	100	MAT C = A–B	10-3
MAT-moltiplicazione *	30	MAT D = A*B	10-3
MAT-moltiplicazione scalare	40	MAT A = (K)*B	10-3
MAT-più +	50	MAT C = C+B	10-3
MAT PRINT	90	MAT PRINT B	10-2
MAT READ	20	MAT READ A(3,4)	10-1
MAT SIZ	30	MAT SIZ A(5,6)	10-4
MAT-TRN	50	MAT B = TRN(X)	10-4
MAT-uguale =	15	MAT A = B	10-3
MAT-ZER	100	MAT A = ZER(M,P)	10-4
Meno –	150	LET A = A–C	2-1
Minore di <	90	IF A < B THEN 350	4-2
Minore o uguale a <=	150	IF A <= B THEN 20	4-2
Moltiplicazione *	40	LET X = Y*Z	2-1
NEXT	120	NEXT K	5-1
ON-GO TO	30	ON K GO TO 10,20,50,90	4-6
ON-THEN	30	ON K THEN 10,20,50,90	4-6
Parentesi ()	50	LET X = (A+B)*(C+D)	2-4
Più +	30	LET A = B+C	2-1
PRINT	40	PRINT "A=",X,"IL NOME E'",N$	3-4
PRINT	80	PRINT # 2,N,A,B$	11-2
PRINTUSING	100	PRINTUSING 110,A,B,X	3-6
QUOTE	40	QUOTE # 2	11-2
RANDOMIZE	30	RANDOMIZE	7-3
READ	20	READ A,B$,C,D	3-1
READ	50	READ:1,A$	11-4
REM	10	REM PROGRAMMA DI MEDIA	1-2
RESTORE	30	RESTORE	3-2
RESTORE$	40	RESTORE$	8-1
RESTORE*	20	RESTORE*	8-1
RETURN	90	RETURN	7-4
SCRATCH	20	SCRATCH # 2	11-2
SET	50	SET:2,P	11-5
SETDIGITS	50	SETDIGITS(4)	3-8
Sottrazione –	20	LET A = B–C	2-1

Termine	Esempio		Par.
STOP	90	STOP	4-3
Uguale =	30	LET A = B	2-5
Uguale =	40	IF A = B THEN 150	4-2
Vettore	30	LET A(I) = X	6-1
WRITE	150	WRITE:2,A$	11-3

APPENDICE B

LE FUNZIONI DI LIBRERIA BASIC

In quest'Appendice diamo un elenco delle funzioni di libreria BASIC. Per la maggior parte sono disponibili su tutti i calcolatori che lavorano in BASIC.

Funzione		**Esempio**		**Par.**
ABS	valore assoluto	10	LET Y = ABS(X)	7-3
ATN	arcotangente = tg^{-1}	30	LET B = ATN(C)	7-3
ASC	valore del simbolo ASCII	50	LET C = ASC(B)	8-3
CHR$	converte un numero o una variabile nel simbolo ASCII	50	LET A$ = CHR$(B)	8-3
CLG	$\log_{10}$	30	LET D = CLG(X)	7-3
COS	coseno - cos	90	LET Y = COS(T)	7-3
COT	cotangente - ctg	150	LET C = COT(B)	7-3
DET	determinante	40	LET A = DET	10-4
EXP	elevamento alla potenza e	20	LET B = EXP(C+1)	7-3
INT	parte intera di un numero	90	LET A = INT(2*B/3)	7-3
LOC	posizione del puntatore in un archivio casuale	50	LET B = LOC(1)	11-4
LOF	posizione della fine di un archivio casuale	100	LET F = LOF(1)	11-4
LGT	$\log_{10}$	50	LET A = LGT(B)	7-3
LOG	logaritmo naturale	20	LET Y = LOG(A)	7-3
NUM	numero di elementi introdotti in un vettore	50	LET B = NUM	10-1
RND	numero casuale	20	LET A = RND	7-3
SGN	+1, −1 o 0, a seconda della funzione	50	LET B = SGN(X)	7-3
SIN	seno — sen	30	LET Y = SIN(T)	7-3
SQR	radice quadrata	70	LET C = SQR(A)	7-3
TAB	specifica le colonne per la stampa	50	PRINT X1,TAB(40);X2	3-7
TAN	tangente — tg	20	LET Y = TAN(Z)	7-3

APPENDICE C

I COMANDI DI SISTEMA IN BASIC

Diamo l'elenco dei comandi di sistema in BASIC, che si usano per gli archivi, per l'esecuzione di un programma, etc. Le modalità operative del sistema BASIC sono spiegate nel paragrafo 1-4. Per i comandi che non sono stati trattati in quella sede diamo qui il paragrafo a cui potrete riferirvi.

Comando	**Scopo**
BYE	Pone termine alla sessione di timesharing. L'utilizzatore disattiva il collegamento.
CATALOGUE	Fa stampare una lista di tutti gli archivi presenti.
GOODBYE	Pone termine alla sessione di timesharing. Il collegamento è disattivato.
LIST	Fa stampare l'archivio corrente.
MON	Trasferisce il controllo dalla modalità BASIC al monitor di sistema (non standard).
NEW	Indica che si vuole creare un nuovo archivio.
OLD	Provvede l'accesso ad un archivio già esistente.
RENAME	Permette di dare un nuovo nome all'archivio corrente (v. par. 11-2).
RUN	Fa compilare ed eseguire il programma corrente in BASIC.
SAVE	Fa sì che l'archivio corrente sia memorizzato. Se non si dà il comando, l'archivio può essere distrutto.
SCRATCH	Fa sì che l'archivio corrente sia cancellato (v. par. 11-2).
SYSTEM	Trasferisce il controllo dalla modalità BASIC al monitor di sistema.
UNSAVE	Elimina la protezione dell'archivio richiesta col comando SAVE.

APPENDICE D

ALCUNE VARIANTI DEL BASIC

In varie installazioni di calcolatori sono state realizzate modifiche del BASIC allo scopo di rendere il linguaggio più economico. In quest'Appendice ci proponiamo di commentare brevemente alcune di queste varianti, che sono presenti in alcune installazioni. Non intendiamo fare qui un discorso completo, ma semplicemente segnalare i tipi di modifiche che vengono effettuate comunemente.

I valori alle variabili vengono assegnati normalmente con lo statement di LET. Ad esempio,

```
30   LET B = A+3
```

Con alcuni compilatori BASIC si può omettere la parola LET; ad esempio si può scrivere:

```
30   B = A+3
```

Alcuni compilatori inoltre permettono di scrivere più assegnazioni sulla stessa linea, divise da virgole. Ad esempio,

```
30   B = A+3,C = D+A↑2,F = G+2                    (D-1)
```

è sostanzialmente equivalente agli statements

```
30   LET B = A+3
31   LET C = D+A↑2
32   LET F = G+2
```

(Si tenga presente che, usando lo statement (D-1), non ci si può riferire al secondo ed al terzo statement con il numero di statement.)

Un'altra variante può aver luogo nello statement di DATA: alcune installazioni permettono di omettere la parola DATA; ad esempio, invece di

```
90   DATA 30 ROSSI,100
```

si può scrivere

```
90   30,ROSSI,100
```

In linea generale, dopo il numero di statement deve comparire uno spazio; se la prima variabile presente nello statement di DATA è una variabile-stringa, dev'essere posta fra virgolette. Ad esempio,

```
100   "ROSSI",90,BIANCO,80
```

Solo la prima variabile-stringa deve necessariamente essere fra virgolette; chiaramente, possono esserlo anche le altre.

Abbiamo segnalato soltanto due varianti fra quelle che semplificano il linguaggio BASIC; altre interverranno caso per caso. Generalmente, queste varianti non soppiantano gli statements originari del BASIC.

L. 17.500 Cod. 525A ISBN88-7056-136-4

Questo libro, propedeutico al linguaggio di programmazione BASIC, è destinato a tutti coloro che non hanno ancora familiarità con la programmazione e, più generalmente, con i calcolatori. Gli argomenti sono esposti in forma elementare ma esauriente, in modo che il lettore alle prime armi, senza trovarsi in difficoltà, possa acquisire familiarità con tutti gli aspetti del BASIC e, da subito, cominciare ad eseguire programmi acquistando, quindi, un'esperienza pratica che lo agevolerà nell'apprendimento.
Oltre ad esporre le caratteristiche del linguaggio, sono esaminate anche la messa a punto dei programmi (debugging), e le procedure necessarie per eseguire programmi: si fa riferimento perciò all'elaborazione a divisione di tempo (*timesharing*) e a quella a lotti (*batch processing*).
Ci auguriamo che questo libro sia di aiuto anche per quei programmatori esperti che vogliono riesaminare questo o quell'aspetto del BASIC. Per facilitare la loro ricerca, infatti, le Appendici presentano un glossario dei termini, delle espressioni e delle operazioni del BASIC, un elenco delle funzioni di libreria, i comandi di sistema, e alcune modifiche del linguaggio.